



**Oracle Database 19c
To Amazon Aurora with MySQL Compatibility (5.7)**

Migration Playbook

2, April 2021

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Table of Contents

Introduction	9
Tables of Feature Compatibility	12
What's New	21
AWS Schema and Data Migration Tools	23
AWS Schema Conversion Tool (SCT)	24
SCT Action Code Index	33
AWS Database Migration Service (DMS)	47
Amazon RDS on Outposts	49
Amazon RDS Proxy	50
Amazon Aurora Serverless v1	52
Amazon Aurora for MySQL Parallel Query	57
Amazon Aurora Backtrack	58
Migration Quick Tips	62
SQL & PL/SQL	64
Oracle Single-Row and Aggregate Functions vs. MySQL Single-Row and Aggregate Functions	65
MySQL Usage	65
Oracle Create Table as Select (CTAS) vs. MySQL Create Table as Select (CTAS)	71
MySQL Usage	71
Oracle Common Table Expression (CTE) vs. MySQL Common Table Expressions (CTE)	73
MySQL Usage	73
Oracle Sequences and Identity Columns vs. MySQL Sequences and AUTO INCREMENT Columns	78
MySQL Usage	81
Oracle Insert From Select vs. MySQL Insert From Select	85
MySQL Usage	86
Oracle Multi-Version Concurrency Control (MVCC) vs. MySQL Multi-Version Concurrency Control (MVCC)	88
MySQL Usage	91
Oracle Merge vs. MySQL Merge	95
MySQL Usage	96
Oracle OLAP Functions vs. MySQL Window Functions	99
MySQL Usage	99
Oracle Transaction Model vs. MySQL Transactions	101

MySQL Usage	103
Oracle Anonymous Block vs. MySQL Transactions or Procedures	107
MySQL Usage	108
Oracle Conversion functions vs. MySQL Conversion Functions	109
MySQL Usage	111
Oracle PL/SQL Cursors vs. MySQL Cursors	112
MySQL Usage	113
Oracle DBMS_DATAPUMP vs. MySQL Integration with Amazon S3	118
MySQL Usage	120
Oracle DBMS_OUTPUT vs. MySQL SELECT	122
MySQL Usage	122
Oracle DBMS_RANDOM vs. MySQL RAND Function	124
MySQL Usage	125
Oracle DBMS_REDEFINITION vs. MySQL Tables and Triggers	127
MySQL Usage	128
Oracle DBMS_SQL Package (11g & 12c) vs. MySQL DBMS_SQL Package	129
MySQL Usage	130
Oracle EXECUTE IMMEDIATE vs. MySQL Execute and Prepare	131
MySQL Usage	132
Oracle Procedures and Functions vs. MySQL Stored Procedures	134
MySQL Usage	137
Oracle Regular Expressions vs. MySQL Regular Expressions	140
MySQL Usage	141
Oracle TIMEZONE Data Type and Functions vs. MySQL CONVERT_TZ Function	144
MySQL Usage	145
Oracle User-Defined Functions vs. MySQL User-Defined Functions	147
MySQL Usage	148
Oracle UTL_FILE vs. MySQL Integration with Amazon S3	150
MySQL Usage	151
Oracle UTL_MAIL & UTL_SMTP(11g & 12c) vs. MySQL Amazon RDS and Amazon Simple Notifi- cation Service (SNS)	155
MySQL Usage	156
Tables & Indexes	162

Case Sensitivity Differences for Oracle and Aurora MySQL	163
Oracle Common Data Types vs. MySQL Common Data Types	164
MySQL Usage	167
Oracle Read-Only Tables and Partitions vs. MySQL Aurora Replicas	172
MySQL Usage	173
Oracle Table Constraints vs. MySQL Table Constraints	175
MySQL Usage	181
Oracle Temporary Tables vs. MySQL Temporary Tables	187
MySQL Usage	189
Oracle Triggers vs. MySQL Triggers	192
MySQL Usage	193
Oracle Tablespaces and Data Files vs. MySQL Tablespaces and Data files	197
MySQL Usage	198
Oracle User-Defined Types vs. MySQL User-Defined Types	201
MySQL Usage	202
Oracle Unused Columns vs. MySQL Unused Columns	203
MySQL Usage	203
Oracle Virtual Columns vs. MySQL Generated Columns	204
MySQL Usage	205
Overall Indexes Summary	207
Oracle Bitmap Indexes vs. MySQL Bitmap Indexes	210
MySQL Usage	210
Oracle B-Tree Indexes vs. MySQL B-Tree Indexes	211
MySQL Usage	211
Oracle Composite Indexes vs. MySQL Multiple-Column Indexes	213
MySQL Usage	213
Oracle Function-Based Indexes vs. MySQL Indexing on Generated Columns	215
MySQL Usage	215
Oracle Invisible Indexes vs. MySQL Invisible Indexes	219
MySQL Usage	220
Oracle Index-Organized Table (IOT) vs. MySQL InnoDB Clustered Index	221
MySQL Usage	222
Oracle Local and Global Partitioned Indexes vs. MySQL Partitioned Indexes	223

MySQL Usage	224
Automatic Indexing	225
MySQL Usage	226
Special Features & Future Content	227
Oracle Advanced Queuing vs. MySQL Integration with Lambda	228
MySQL Usage	229
Oracle Character Sets vs. MySQL Character Sets and Collations	230
MySQL Usage	231
Oracle Database Links vs. MySQL Fully-Qualified Table Names	233
MySQL Usage	234
Oracle DBMS_SCHEDULER vs. MySQL Events	235
MySQL Usage	238
Oracle External tables vs. MySQL Integration with Amazon S3	242
MySQL Usage	243
Oracle Inline Views vs. MySQL Inline Views	248
MySQL Usage	248
Oracle JSON Document Support vs. MySQL JSON	249
MySQL Usage	250
Oracle Materialized Views vs. MySQL Summary Tables or Views	253
MySQL Usage	254
Oracle Multitenant vs. MySQL Databases	256
MySQL Usage	259
Oracle Resource Manager vs. MySQL Dedicated Amazon Aurora Clusters	262
MySQL Usage	263
Oracle SecureFile LOBs vs. MySQL Large Objects (LOB)	268
MySQL Usage	268
Oracle Synonyms vs. MySQL Synonyms	270
MySQL Overview	271
Oracle Views vs. MySQL Views	272
MySQL Usage	273
Oracle XML DB vs. MySQL XML	277
MySQL Usage	281
Oracle Table Compression vs. MySQL InnoDB Table Compression	284

MySQL Usage	285
Oracle LogMiner vs. MySQL Logs and the mysqlbinlog Utility	287
MySQL Usage	288
Oracle SQL Result Cache vs. MySQL Query Cache	294
MySQL Usage	295
High Availability and Disaster Recovery (HADR)	296
Oracle Active Data Guard vs. MySQL Replicas	297
MySQL Usage	298
Oracle Real Application Clusters (RAC) vs. MySQL Aurora Architecture	300
MySQL Usage	302
Migrate to Aurora MySQL Serverless	307
Oracle Traffic Director vs. Amazon RDS Proxy for Amazon Aurora MySQL	309
MySQL Usage	309
Oracle Data Pump vs. MySQL mysqldump and mysql	311
MySQL Usage	312
Oracle Flashback Database vs. MySQL Snapshots	315
MySQL Usage	315
Oracle Flashback Table vs. MySQL Snapshots (Flashback Tables)	321
MySQL Usage	322
Oracle Recovery Manager (RMAN) vs. MySQL Snapshots (RMAN)	324
MySQL Usage	325
Oracle SQL*Loader vs. MySQL mysqlimport and LOAD DATA	329
MySQL Usage	330
Configuration	331
Oracle Upgrades vs. Aurora for MySQL Upgrades	332
MySQL Usage	334
Oracle Alert.log and logs vs. MySQL Error Log via Amazon RDS Console	338
MySQL Usage	339
Oracle SGA & PGA Memory Sizing vs. MySQL Memory Buffers	342
MySQL Usage	343
Oracle Instance Parameters vs. MySQL Aurora Parameter Groups	346
MySQL Usage	346
Oracle Session Parameters vs. MySQL Session Variables	350

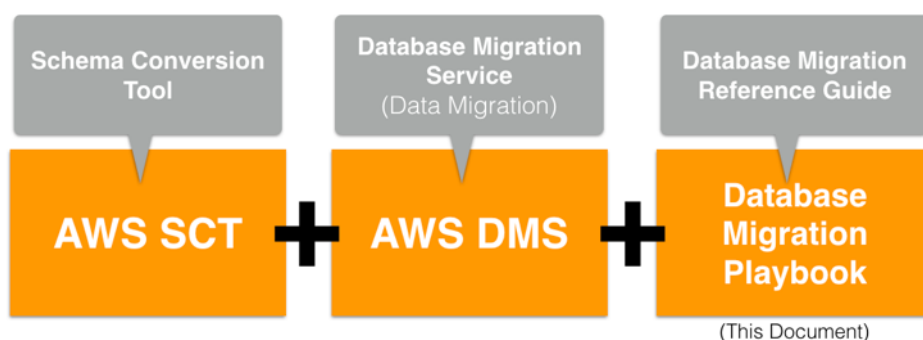
MySQL Usage	351
Performance Tuning	353
Oracle Database Hints vs. MySQL Database Hints	354
MySQL Usage	355
Oracle Execution Plans vs. MySQL Execution Plans	359
MySQL Usage	360
Oracle Table Statistics vs. MySQL Managing Statistics	363
MySQL Usage	364
Security	368
Oracle Encrypted Connections vs. Aurora for MySQL Encrypted Connections	369
MySQL Usage	369
Oracle Transparent Data Encryption (TDE) vs. MySQL AWS Aurora Encryption and Column Encryp- tion	370
MySQL Usage	373
Oracle Roles vs. MySQL Privileges	379
MySQL Usage	380
Oracle Database Users vs. MySQL Users	382
MySQL Usage	383
Physical Storage	386
Oracle Table Partitioning vs. MySQL Table Partitioning	387
MySQL Usage	390
Sharding	395
Migrate to Aurora MySQL	396
Monitoring	397
Oracle V\$ Views and the Data Dictionary vs. MySQL System Catalog & SHOW command	398
MySQL Usage	399
Migration Quick Tips	401
Migration Quick Tips	402
Glossary	404

Introduction

The migration process from a source database (Oracle or SQL Server) to Amazon Aurora (PostgreSQL or MySQL) typically involves several stages. The first stage is to use the AWS Schema Conversion Tool (SCT) and the AWS Database Migration Service (DMS) to convert and migrate the schema and data. While most of the migration work can be automated, some aspects require manual intervention and adjustments to both database schema objects and database code.

In order to provide the complete picture to all users, RDS related information is being mentioned in this playbook as well. Sections formatted in the following way are relevant for RDS only:

RDS MySQL Only (not Aurora MySQL): This paragraph is about the latest db engine version which is supported only in RDS (and not Aurora)



The purpose of this Playbook is to assist administrators tasked with migrating from source databases to Aurora with the aspects that can't be automatically migrated using the Amazon Web Services Schema Conversion Tool (AWS SCT). It focuses on the differences, incompatibilities, and similarities between the source database and Aurora in a wide range of topics including T-SQL, Configuration, High Availability and Disaster Recovery (HADR), Indexing, Management, Performance Tuning, Security, and Physical Storage.

The first section of this document provides [an overview of AWS SCT](#) and the [AWS Data Migration Service \(DMS\)](#) tools for automating the migration of schema, objects and data. The remainder of the document contains individual sections for the source database features and their Aurora counterparts. Each section provides a short overview of the feature, examples, and potential workaround solutions for incompatibilities.

You can use this playbook either as a reference to investigate the individual action codes generated by the AWS SCT tool, or to explore a variety of topics where you expect to have some incompatibility issues. When using the AWS SCT, you may see a report that lists *Action codes*, which indicates some manual conversion is required, or that a manual verification is recommended. For your convenience, this Playbook includes an [SCT Action Code Index](#) section providing direct links to the relevant topics that discuss the manual conversion tasks needed to address these action codes. Alternatively, you can explore the [Tables of Feature Compatibility](#) section that provides high-level graphical indicators and descriptions of the feature compatibility between the source database and Aurora. It also includes a graphical compatibility indicator and links to the actual sections in the playbook.

There is appendix at the end of this playbook: [Appendix: Migration Quick Tips](#) provides a list of tips for administrators or developers who have little experience with Aurora (PostgreSQL or MySQL). It briefly highlights key differences between the source database and Aurora that they are likely to encounter.







Note that not all of the source database features are fully compatible with Aurora or have simple workarounds. From a migration perspective, this document does not yet cover all source database features and capabilities. This first release focuses on some of the most important features and will be expanded over time.

Disclaimer







The various code snippets, commands, guides, best practices, and scripts included in this document should be used for reference only and are provided as-is without warranty. Test all of the code, commands, best practices, and scripts outlined in this document in a non-production environment first. Amazon and its affiliates are not responsible for any direct or indirect damage that may occur from the information contained in this document.

Tables of Feature Compatibility

Feature Compatibility Legend

Compatibility Score Symbol	Description
	Very high compatibility: None or minimal low-risk and low-effort rewrites needed
	High compatibility: Some low-risk rewrites needed, easy workarounds exist for incompatible features
	Medium compatibility: More involved low-medium risk rewrites needed, some redesign may be needed for incompatible features
	Low compatibility: Medium to high risk rewrites needed, some incompatible features require redesign and reasonable-effort workarounds exist
	Very low compatibility: High risk and/or high-effort rewrites needed, some features require redesign and workarounds are challenging
	Not compatible: No practical workarounds yet, may require an application level architectural solution to work around incompatibilities

SCT/DMS Automation Level Legend

SCT/DMS Automation Level Symbol	Description
	Full Automation Perform fully automatic conversion, no manual conversion needed
	High Automation: Minor, simple manual conversions may be needed
	Medium Automation: Low-medium complexity manual conversions may be needed
	Low Automation: Medium-high complexity manual conversions may be needed
	Very Low Automation: High risk or complex manual conversions may be needed
	No Automation: Not currently supported, manual conversion is required for this feature

SQL Language

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Aggregate Functions	Aggregate Functions	<ul style="list-style-type: none"> Not all functions are supported by MySQL and may require manual creation 	
Create Table As select (CTAS)	Create Table As select (CTAS)		
Common Table Expression (CTE)	Common Table Expression (CTE)	<ul style="list-style-type: none"> CTE is not supported in MySQL, workaround available 	
Insert As Select	Insert As Select	<ul style="list-style-type: none"> ERROR LOG and subquery options are not supported in MySQL 	
Locking	Locking	<ul style="list-style-type: none"> ERROR LOG and subquery options are not supported 	
MERGE	MERGE	<ul style="list-style-type: none"> MERGE is not supported in MySQL, workaround available 	
OLAP Functions	OLAP Functions	<ul style="list-style-type: none"> GREATEST and LEAST functions may produce different results in MySQL CONNECT BY is not supported by MySQL, workaround available 	
Sequences and Identity	Sequences and Identity	<ul style="list-style-type: none"> MySQL does not support sequences Identity column has different syntax and options for MySQL 	
Transactions	Transactions	<ul style="list-style-type: none"> MySQL default isolation level is REPEATABLE READ Nested transactions are not supported in MySQL 	

Tables

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Data Types	Data Types	<ul style="list-style-type: none"> BFILE, ROWID, UROWID are not supported by MySQL 	
Read Only	Read Only	<ul style="list-style-type: none"> READ ONLY mode is not supported in MySQL, use a work-around 	
Constraints	Constraints	<ul style="list-style-type: none"> CHECK, REF, DEFERRABLE, and DISABLE constraints are not supported in MySQL Constraints on Views are not supported in MySQL 	
Temp Table	Temp Table	<ul style="list-style-type: none"> GLOBAL temporary table is not supported in MySQL Cannot read from multiple sessions in MySQL Table dropped after session ends in MySQL 	
Triggers	Triggers	<ul style="list-style-type: none"> Statement and system event triggers are not supported in MySQL CREATE OR REPLACE is not supported in MySQL 	
Tablespaces	Tablespaces	<ul style="list-style-type: none"> Tablespace-per-file only and no physical file attributes are allowed in Aurora MySQL 	
User Defined Types	User Defined Types	<ul style="list-style-type: none"> User-Defined Types are not supported in MySQL 	
Unused Columns	Unused Columns	<ul style="list-style-type: none"> MySQL does not support unused columns 	
Virtual Columns	Virtual Columns	<ul style="list-style-type: none"> Different paradigm and syntax in MySQL 	

Configuration

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Alerting	Alerting	<ul style="list-style-type: none"> Use Event Notifications Subscription with Amazon Simple 	

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
		Notification Service (SNS)	
Cache and Pools	Cache and Pools	<ul style="list-style-type: none"> Different cache names, similar usage in MySQL 	
Database Parameters	Database Parameters	<ul style="list-style-type: none"> Use Cluster and Database/Cluster Parameters 	
Session Parameters	Session Parameters	<ul style="list-style-type: none"> SET options are significantly different in MySQL 	

Special Features

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Advanced Queues	Advanced Queues	<ul style="list-style-type: none"> Use AWS Lambda and Amazon SQS with Aurora MySQL 	
Character Set	Character Set	<ul style="list-style-type: none"> Different syntax for MySQL MySQL can have different collation for each database in the same instance 	
Database Links	Database Links	<ul style="list-style-type: none"> MySQL does not support database links 	
DBMS_SCHEDULER	DBMS_SCHEDULER	<ul style="list-style-type: none"> Different paradigm and syntax for MySQL 	
External Tables	External Tables	<ul style="list-style-type: none"> Use Aurora MySQL integration with Amazon S3, different paradigm and syntax 	
Inline Views	Inline Views		
JSON	JSON	<ul style="list-style-type: none"> Different paradigm and syntax in MySQL, requires application/drivers rewrite 	
Materialized Views	Materialized Views	<ul style="list-style-type: none"> MySQL doesn't support a materialized VIEW 	
Multi-tenant	Multi-tenant	<ul style="list-style-type: none"> Distribute load/applications/users across multiple instances 	
Resource Man-	Resource Manager	<ul style="list-style-type: none"> Distribute load/applications/ 	

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
ager		users across multiple instances	
Secure File and LOBs	Secure File and LOBs	<ul style="list-style-type: none"> SecureFiles are not supported in MySQL, automation and compatibility refer only to LOBs 	
Synonyms	Synonyms	<ul style="list-style-type: none"> Use stored procedures and functions to abstract instance-wide objects 	
Views	Views	<ul style="list-style-type: none"> MySQL does not support View with the READ ONLY option MySQL does not support DML on views 	
XML	XML	<ul style="list-style-type: none"> Different paradigm and syntax in MySQL, requires application/drivers rewrite 	

High Availability and Disaster Recovery (HADR)

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Active Data Guard	Active Data Guard	<ul style="list-style-type: none"> Distribute load/ applications/users across multiple instances 	
RAC	RAC	<ul style="list-style-type: none"> Distribute load/ applications/users across multiple instances 	

Indexes

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
BITMAP	BITMAP	<ul style="list-style-type: none"> MySQL does not support BITMAP index 	
BTree	BTree		

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Composite	Composite		
Function Based Index (FBI)	Function Based Index (FBI)	<ul style="list-style-type: none"> MySQL doesn't support functional indexes; a workaround is available 	
Invisible Indexes	Invisible Indexes	<ul style="list-style-type: none"> MySQL does not support invisible indexes 	
Index-Organized Table (IOT)	Index-Organized Table (IOT)	<ul style="list-style-type: none"> MySQL does not support the index-organized tables object due to this is the default behavior for InnoDB 	
Local and Global Indexes	Local and Global Indexes	<ul style="list-style-type: none"> MySQL does not support global indexes 	

PL/SQL

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Anonymous Block	Anonymous Block	<ul style="list-style-type: none"> Different syntax in MySQL may require code rewrite 	
Conversion Functions	Conversion Functions	<ul style="list-style-type: none"> Not all functions are supported by MySQL and may require manual creation 	
Cursors	Cursors	<ul style="list-style-type: none"> Minor differences in syntax in MySQL, may require some code rewrite. %ISOPEN, %ROWTYPE, and %BULK_ROWCOUNT are not supported by MySQL 	
DBMS_DATAPUMP	DBMS_DATAPUMP	<ul style="list-style-type: none"> No equivalent tool with MySQL 	
DBMS_OUTPUT	DBMS_OUTPUT	<ul style="list-style-type: none"> Different paradigm and syntax requires application/drivers rewrite. 	

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
DBMS_RANDOM	DBMS_RANDOM	<ul style="list-style-type: none"> Different syntax and missing options in MySQL, may require code rewrite 	
DBMS_REDEFINITION	DBMS_REDEFINITION	<ul style="list-style-type: none"> MySQL does not support DBMS_REDEFINITION 	
DBMS_SQL	DBMS_SQL	<ul style="list-style-type: none"> Different paradigm and syntax in MySQL, requires application/drivers rewrite. 	
EXECUTE IMMEDIATE	EXECUTE IMMEDIATE	<ul style="list-style-type: none"> Must use PREPARE command in MySQL Execute SQL with results and bind variables, or Anonymous block execution using EXECUTE, are not supported in MySQL 	
Procedures and Functions	Procedures and Functions	<ul style="list-style-type: none"> Syntax and option differences in MySQL 	
Regular Expressions	Regular Expressions	<ul style="list-style-type: none"> Syntax and option differences in MySQL 	
TIMEZONE data type and functions	TIMEZONE data type and functions	<ul style="list-style-type: none"> There is no equivalent option for Oracle's CREATE TABLE...TIMESTAMP WITH TIME ZONE in MySQL but CONVERT_TZ can be used to achieve the same results 	
User Defined Functions	User Defined Functions	<ul style="list-style-type: none"> Syntax and option differences in MySQL 	
UTL_FILE	UTL_FILE	<ul style="list-style-type: none"> MySQL does not support UTL_FILE but Aurora MySQL has a built-in integration with S3 	
UTL_MAIL	UTL_MAIL & UTL	<ul style="list-style-type: none"> Use AWS Lambda Integration 	

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
& UTL_SMTP	SMTP	with MySQL	

Performance Tuning

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Database Hints	Database Hints	<ul style="list-style-type: none"> Very limited set of hints in MySQL, Index hints, and optimizer hints as comments in MySQL Syntax differences in MySQL 	
Execution Plan	Execution Plan	<ul style="list-style-type: none"> Syntax differences in MySQL Completely different optimizer with different operators and rules in MySQL 	
Statistics	Statistics	<ul style="list-style-type: none"> Syntax and option differences in MySQL, similar functionality 	

Physical Storage

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Partitions	Partitions	<ul style="list-style-type: none"> Interval Partitioning, Partition Advisor, Preference Partitioning, Virtual Column Based Partitioning, and Automatic List Partitioning are not supported by MySQL 	

Security

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Encryption	Encryption	<ul style="list-style-type: none"> Use Amazon Aurora Encryption 	
Roles	Roles	<ul style="list-style-type: none"> There are no roles, only privileges in MySQL 	
Users	Users	<ul style="list-style-type: none"> Syntax and option differences, similar functionality 	

Future content

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Compression	Compression	<ul style="list-style-type: none"> Syntax and option differences in MySQL, similar functionality Compress a partition are not supported in MySQL 	

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
LogMiner	LogMiner	<ul style="list-style-type: none"> MySQL does not support LogMiner, workaround available 	
Result Cache	Result Cache	<ul style="list-style-type: none"> Syntax and option differences in MySQL, similar functionality Recommend do not use 	

Backup and Recovery

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
DataPump	DataPump	<ul style="list-style-type: none"> Not compatible with MySQL 	
Flashback Database	Flashback Database	<ul style="list-style-type: none"> Storage-level backup managed by Amazon RDS 	
Flashback Table	Flashback Table	<ul style="list-style-type: none"> Storage-level backup managed by Amazon RDS 	
RMAN	RMAN	<ul style="list-style-type: none"> Storage-level backup managed by Amazon RDS 	
SQL*Loader	SQL*Loader	<ul style="list-style-type: none"> Not compatible with MySQL 	

Monitoring

Oracle	Aurora MySQL 5.7	Key Differences	Compatibility
Information Views	Information Views	<ul style="list-style-type: none"> Table names in queries must be changed when using MySQL 	

What's New

The Aurora MySQL compatibility database version 5.7 hasn't changed since the last version of this playbook, however, AWS developed several new features that can help maintain and improve performance.

Also, this updated version of the book will mention newer features of Oracle and what can be done with those in MySQL and notes for MySQL 8.0 when higher compatibility level can be achieved.

As of the time of this playbook release, Aurora does not offer MySQL 8.0 yet, however, RDS MySQL does support MySQL 8.0. MySQL 8.0 offers certain feature enhancements over the previous major versions of MySQL, such as Common Table Expressions, Windowing Functions, and role based security. These newer features in MySQL 8.0 can provide better compatibility with SQL Server. This document will call out where MySQL 8.0 improves compatibility with SQL Server, in case there is interest in exploring RDS MySQL 8.0 as a target database as well.

Update	Aspect	Link
Updated screen shots	AWS	AWS SCT
Updated SCT warning lists	AWS	AWS SCT Error Codes
Amazon RDS on Outposts	AWS	RDS Outposts
Amazon RDS Proxy	AWS	RDS Proxy
Amazon Aurora Serverless	AWS	RDS Serverless
AWS RDS Backtrack	AWS	RDS Backtrack
AWS RDS Parallel Query	AWS	RDS Parallel Query
Online partition merging and partition strategy modification	Oracle	Partitions
Oracle 19 Hybrid partitioned tables		
New partitions options	MySQL	Partitions
Unicode 9.0	Oracle	Character Set
Oracle private temporary tables	Oracle	Temporary Tables
New options of temporary tables	MySQL	Temporary Tables
Oracle sharding	Oracle	Sharding
Inline external tables	Oracle	External Tables
Traffic Director Mode of Connection Manager	Oracle	Connection Manager
New Oracle (18c + 19c) PDB / CDB features like: CDB fleets, DBCA PDB Clone, PDB Clone switcher and more	Oracle	Container Databases
Scalable sequences	Oracle	Sequences
New Oracle 19 function, JSON_SERIALIZE	Oracle	JSON
New JSON options and functions	MySQL	JSON
New Oracle 19 adding DISTINCT option for LISTAGG function and new bitmap functions to speed up COUNT DISTINCT	Oracle	Aggregate Functions
New topic of Automatic Indexing	Oracle	Automatic Indexing
Oracle 19 real-time statistics collection and High-Frequency	Oracle	Statistics

Update	Aspect	Link
Automatic Optimizer Statistics Collection		
INNODB_CACHED_INDEXES	MySQL	Statistics
Oracle 19 SQL Quarantine	Oracle	Execution Plans
Case sensitive topic to call out the difference between the two engines	MySQL	Case Sensitive
Roles support and user account categories	MySQL	Roles
FIPS support schema and tablespace encryption	MySQL	Encryption
innodb_deadlock_detect NOWAIT and SKIP LOCKED	MySQL	Transactions
Disabling redo logging	MySQL	Backup and recovery
Descending and invisible indexes	MySQL	Invisible Indexes
CTE support	MySQL	CTE
window Functions support	MySQL	window Functions
EXPLAIN ANALYZE	MySQL	Execution Plan
Encrypted Connections	New topic	Encrypted Connections
Upgrades	New topic	Upgrades
Amazon RDS Proxy	New topic	Connection Manager MySQL

General migration tips topic has been added here: [link](#)

Use the following links to find out about other new Aurora specific features [AWS Aurora News](#).

In order to provide the complete picture to all users, RDS related information is being mentioned in this playbook as well. Sections formatted in the following way are relevant for RDS only:

RDS MySQL Only (not Aurora MySQL): This paragraph is about the latest db engine version which is supported only in RDS (and not Aurora)

AWS Schema and Data Migration Tools



AWS Schema Conversion Tool (SCT)

Usage

The AWS Schema Conversion Tool (SCT) is a Java utility that connects to source and target databases, scans the source database schema objects (tables, views, indexes, procedures, etc.), and converts them to target database objects.

This section provides a step-by-step process for using AWS SCT to migrate an Oracle database to an Aurora PostgreSQL database cluster. Since AWS SCT can automatically migrate most of the database objects, it greatly reduces manual effort.

It is recommended to start every migration with the process outlined in this section and then use the rest of the Playbook to further explore manual solutions for objects that could not be migrated automatically. For more information, see

<http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html>

Migrating a Database

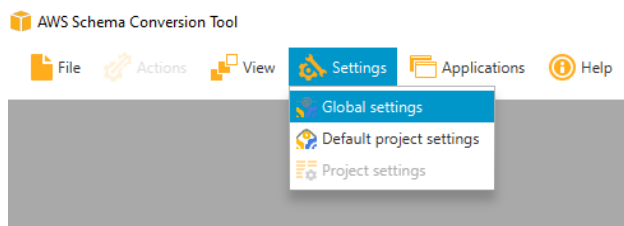
Note: This walkthrough uses the AWS DMS Sample Database. You can download it from <https://github.com/aws-samples/aws-database-migration-samples>.

Download the Software and Drivers

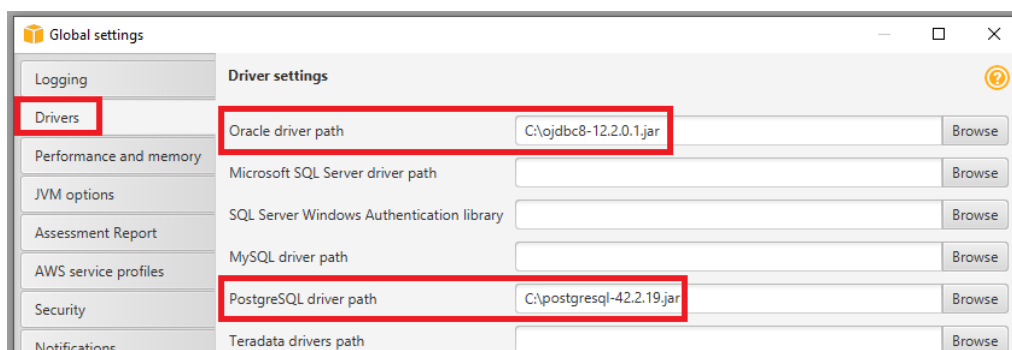
1. Download and install the AWS SCT from https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Installing.html.
2. Download the relevant drivers:
 - Oracle
<http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html>
 - SQL Server
<https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774>
 - PostgreSQL
<https://jdbc.postgresql.org/>
 - MySQL
<https://www.mysql.com/products/connector/>
 - Other link to supported drivers can be found in here:
https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Installing.html#CHAP_Installing.JDBCDrivers

Configure SCT

Launch SCT. Click the **Settings** button and select **Global Settings**.

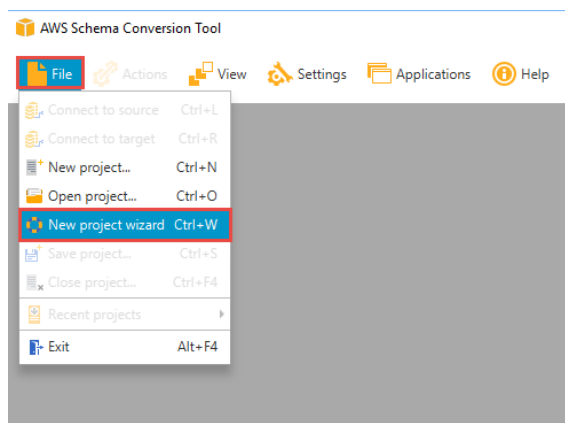


On the left navigation bar, click **Drivers**. Enter the paths for the Oracle and PostgreSQL drivers downloaded in the first step. Click **Apply** and then **OK**.



Create a New Migration Project

Click **File > New project wizard**. Alternatively, use the keyboard shortcut **<Ctrl+W>**.



Enter a project name and select a location for the project files. Click **Next**.

Create a new database migration project

The AWS Schema Conversion Tool can help migrate your database to the database platform of your choice. Specify the database to migrate to AWS.

Step 1. Choose a source

Project name:

Location:

Transactional database (OLTP)
 Data warehouse (OLAP)
 NoSQL database
 ETL

Source engine:

I want to switch engines and optimize for the cloud
 I want to keep the same engine but optimize for the cloud
 I want to see a combined report for database engine switch and optimization to cloud

Enter connection details for the source Oracle database and click **Test Connection** to verify. Click **Next**.

Create a new database migration project

Specify information about the source database to connect to.

Note: The AWS Schema Conversion Tool doesn't store the password. If you close your AWS Schema Conversion Tool project and reopen it, you are prompted for the password to connect your source database as needed.

Connect to Oracle

Connection:

Type:

Server name:

Server port:

Oracle SID:

User name:

Password:

Use SSL
 Store password

Select the schema or database to migrate and click **Next**.

Create a new database migration project

Step 1. Choose a source

Step 2. Connect to the source database

Step 3. Choose a schema

Step 4. Run the database migration assessment

Step 5. Choose a target

Choose the schema that you want to analyze.

- ▼ postgres@oraplaybook.crv77o85iv8n.eu-central-1.rds.amazonaws.com:1521:ORCL
 - ▼ Schemas [24]
 - ANONYMOUS
 - APPQOSSYS
 - AUDSYS
 - CTXSYS
 - DBSFUSER
 - DBSNMP
 - DIP
 - DMS_SAMPLE
 - GSMADMIN_INTERNAL
 - GSMCATUSER
 - GSMUSER

The progress bar displays the objects being analyzed.

Oracle database

Loading schema dependencies...

0% [Abort](#)

The Database Migration Assessment Report is displayed when the analysis completes. Read the Executive summary and other sections. Note that the information on the screen is only partial. To read the full report, including details of the individual issues, click **Save to PDF** and open the PDF document.

Step 1. Choose a source

Step 2. Connect to the source database

Step 3. Choose a schema

Step 4. Run the database migration assessment

Step 5. Choose a target

[Save to CSV](#) [Save to PDF](#)

Database Switch Assessment

Executive summary

Target platform	Auto or minimal changes			Complex actions			
	Storage objects	Code objects	Conversion actions	Storage objects		Code objects	
				Objects count	Conversion actions	Objects count	Conversion actions
Amazon RDS for MySQL	63 (100%)	12 (52%)	12	0 (0%)	0	11 (48%)	55
Amazon Aurora (MySQL compatible)	61 (97%)	12 (52%)	15	2 (3%)	0	11 (48%)	55
Amazon RDS for PostgreSQL	63 (100%)	20 (87%)	11	0 (0%)	0	3 (13%)	1
Amazon Aurora (PostgreSQL compatible)	63 (100%)	20 (87%)	11	0 (0%)	0	3 (13%)	1
Amazon RDS for MariaDB	61 (97%)	13 (57%)	20	2 (3%)	0	10 (43%)	50

We completed the analysis of your Oracle source database and estimate that 100% of the database storage objects and 52% of database code objects can be converted automatically or with minimal changes if you select Amazon RDS for MySQL as your migration target. Database storage objects include schemas, tables, table constraints, indexes, types, collection types, sequences, synonyms, view-constraints, clusters and database links. Database code objects include triggers, views, materialized views, materialized view logs, procedures, functions, packages, package constants, package cursors, package exceptions, package variables, package functions, package procedures, package types, package collection types, scheduler-jobs, scheduler-programs, scheduler-schedules and queueing-tables. Based on the source code syntax analysis, we estimate 90% (based on #

[Previous](#) [Next](#) [Cancel](#)

Scroll down to the section **Database objects with conversion actions for Amazon Aurora (PostgreSQL compatible)**.

Of the total 63 database storage object(s) and 23 database code object(s) in the source database, we identified 63 (100%) database storage object(s) and 20 (87%) database code object(s) that can be converted to Amazon Aurora (PostgreSQL compatible) automatically or with minimal changes.

3 (13%) database code object(s) require 1 complex user action(s) to complete the conversion.

Figure: Conversion statistics for database storage objects

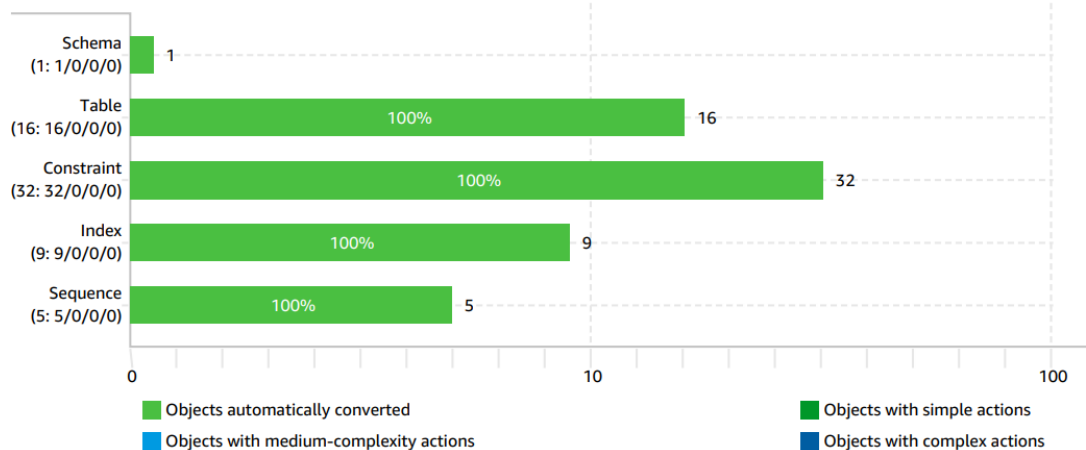
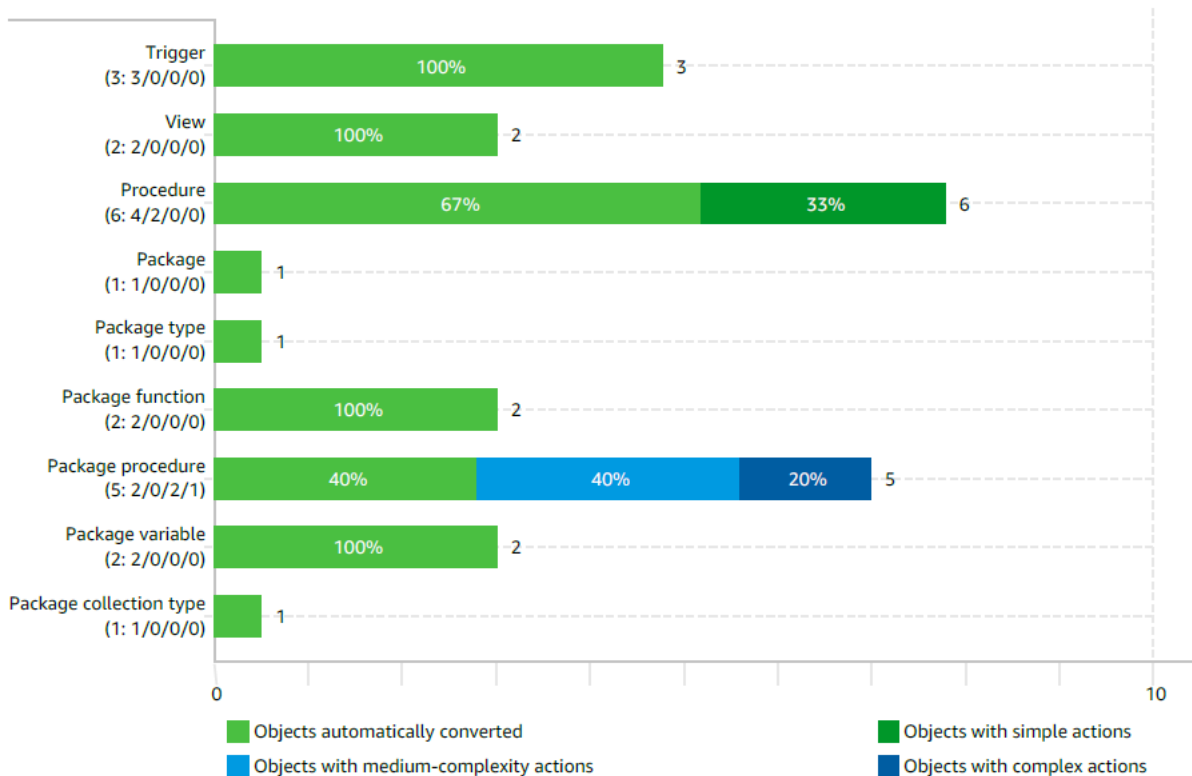
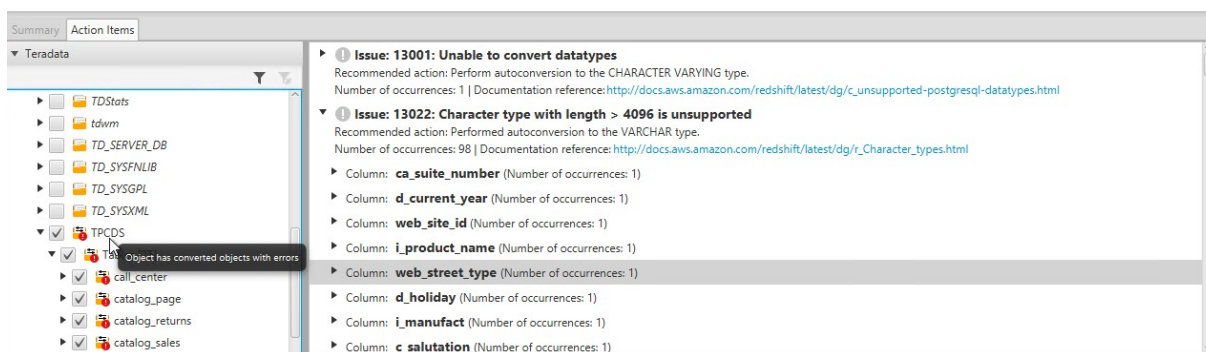


Figure: Conversion statistics for database code objects

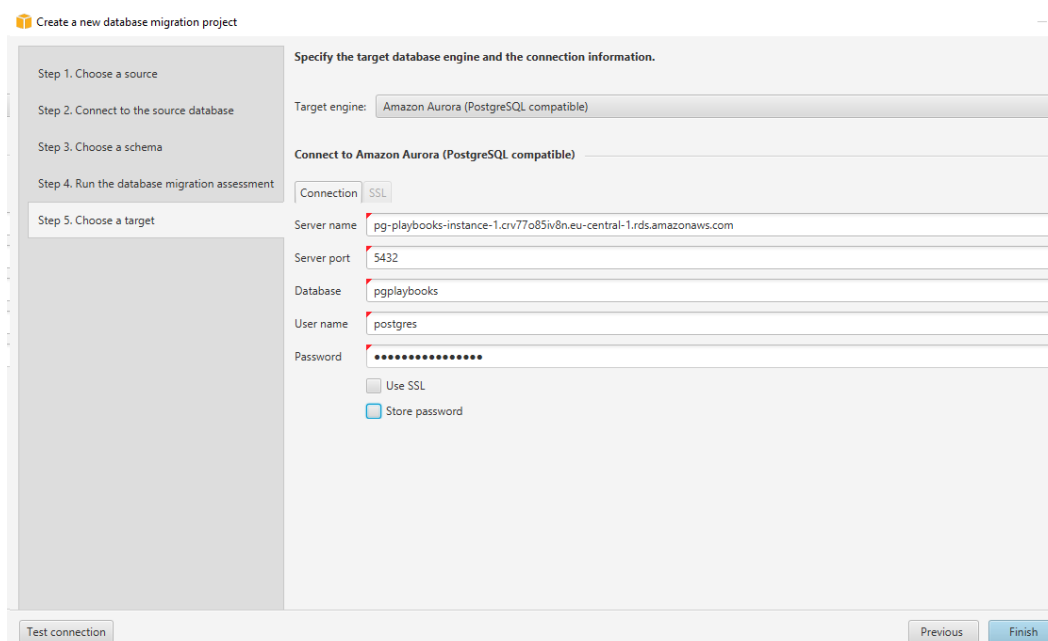


Scroll further down to the section **Detailed recommendations for Amazon Aurora (PostgreSQL compatible) migrations**.



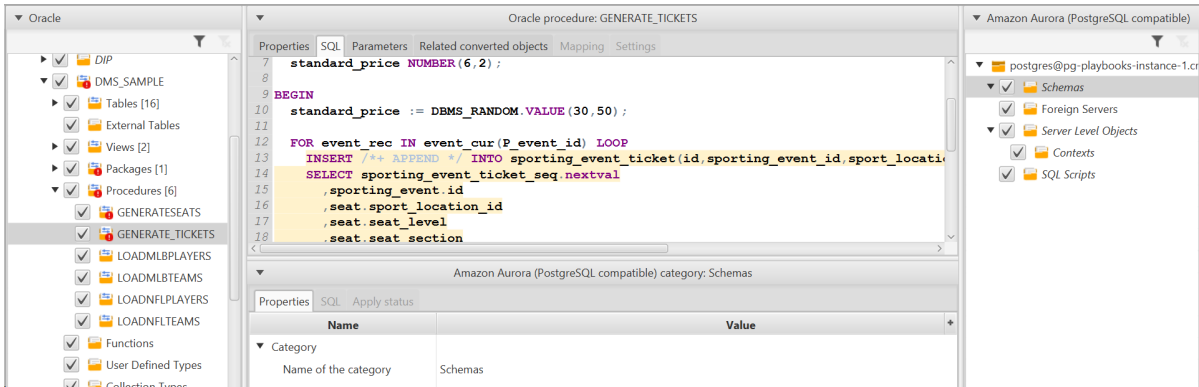
Return to AWS SCT and click **Next**. Enter the connection details for the target Aurora PostgreSQL database and click **Finish**.

Note: The changes have not yet been saved to the target.

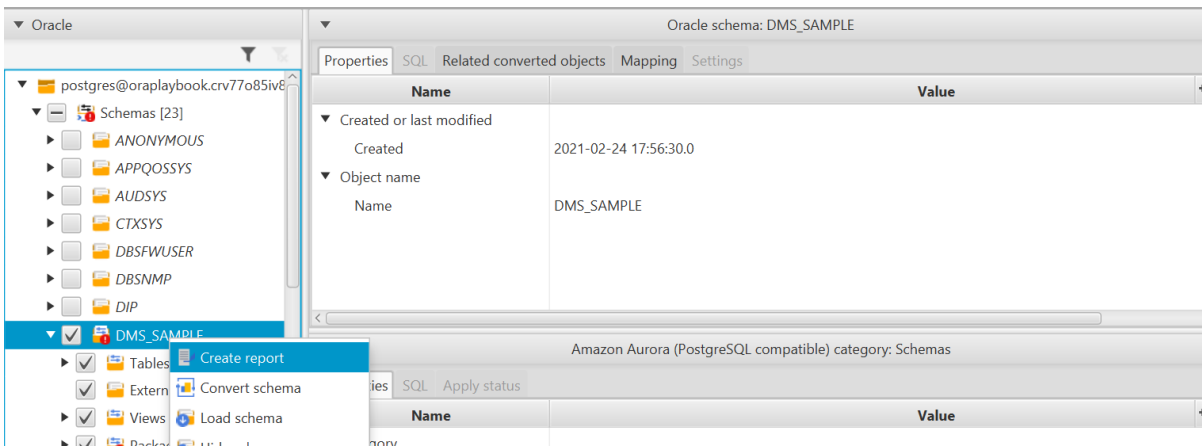


When the connection is complete, AWS SCT displays the main window. In this interface, you can explore the individual issues and recommendations discovered by AWS SCT.

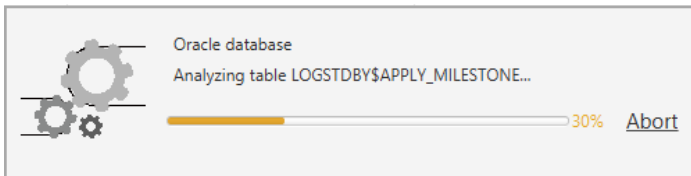
For example, expand **sample database > dms sample > Proceduress > generate_tickets**. This issue has a red marker indicating it could not be automatically converted and requires a manual code change (issue 811 above). Select the object to highlight the incompatible code section.



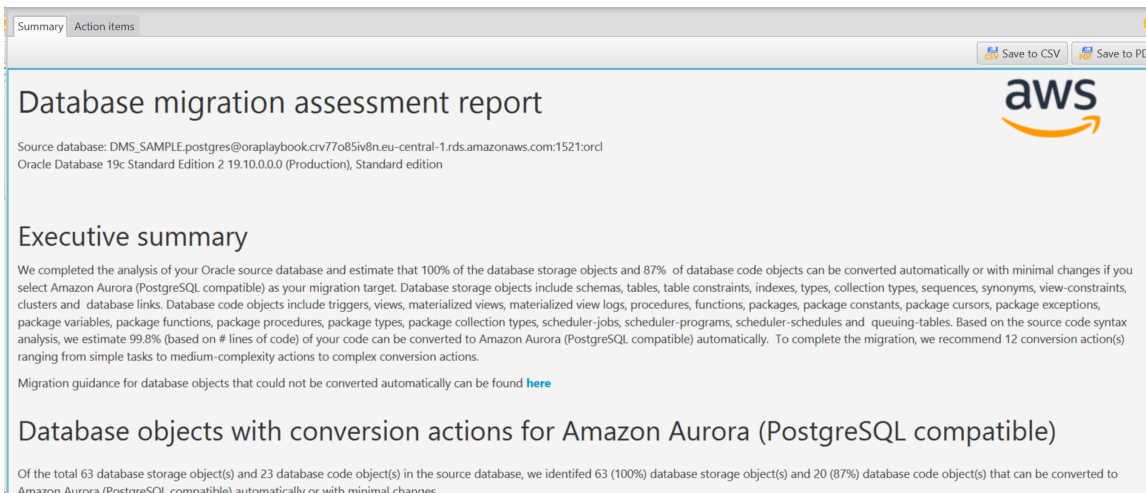
Right-click the schema and then click **Create Report** to create a report tailored for the target database type. It can be viewed in AWS SCT.



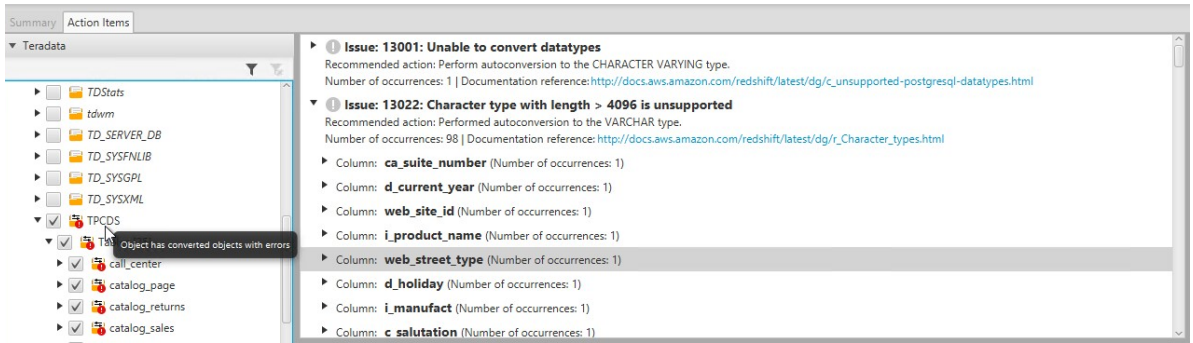
The progress bar updates while the report is generated.



The executive summary page displays. Click the **Action Items** tab.



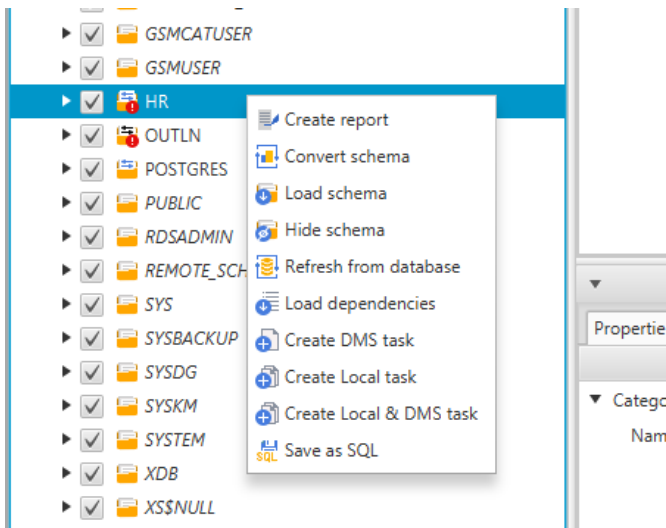
In this window, you can investigate each issue in detail and view the suggested course of action. For each issue, drill down to view all instances of that issue.



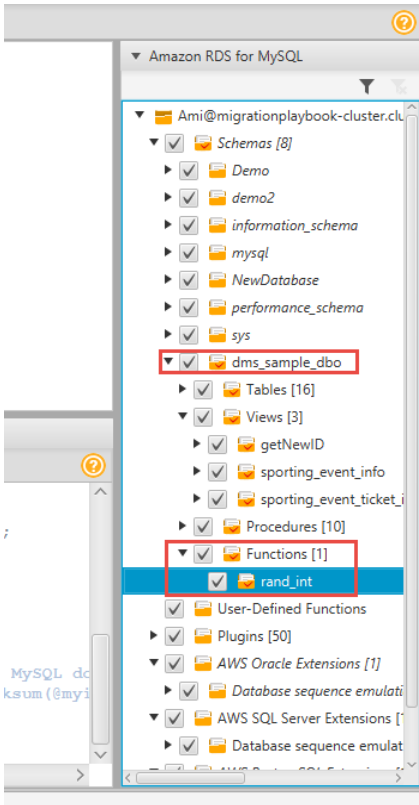
Right-click the database name and click **Convert Schema**.

Be sure to uncheck the **sys** and **information_schema** system schemas. Aurora PostgreSQL already has an **information_schema** schema.

This step does not make any changes to the target database.

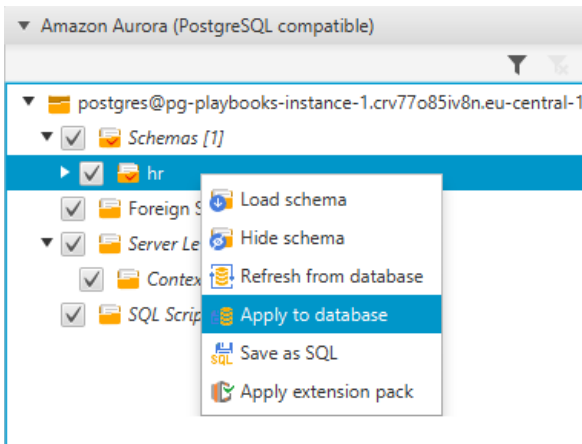


On the right pane, the new virtual schema is displayed as if it exists in the target database. Drilling down into individual objects displays the actual syntax generated by AWS SCT to migrate the objects.



Right-click the database on the right pane and choose either **Apply to database** to automatically execute the conversion script against the target database, or click **Save as SQL** to save to an SQL file.







Saving to an SQL file is recommended because it allows you to verify and QA the SCT code. Also, you can make the adjustments needed for objects that could not be automatically converted.



For more information, see https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Welcome.html

SCT Action Code Index

Legend

SCT/DMS Automation Level Symbol	Description
	Full Automation SCT performs fully automatic conversion, no manual conversion needed.
	High Automation: Minor, simple manual conversions may be needed.
	Medium Automation: Low-medium complexity manual conversions may be needed.
	Low Automation: Medium-high complexity manual conversions may be needed.
	Very Low Automation: High risk or complex manual conversions may be needed.
	No Automation: Not currently supported by SCT, manual conversion is required for this feature.

The following sections list the [Schema Conversion Tool](#) Action codes for topics covered in this playbook.

Note: The links in each section point to the Microsoft Oracle topic pages, which are immediately followed by the MySQL pages for the same topic.

Creating Tables



AWS SCT automatically converts the most commonly used constructs of the CREATE TABLE statement as both Oracle and Aurora MySQL support entry level ANSI compliance. These items include table names, security schema, column names, basic column data types, column constraints, table constraints, column default values, primary keys, candidate (UNIQUE) keys, and foreign keys. Some changes may be required for computed columns and global temporary tables.

Action Code	Action Message
73	MySQL doesn't support the IDENTITY statement with the MAXVALUE, MINVALUE, CYCLE option and when INCREMENT BY value different from 1.
74	MySQL doesn't support the AUTO_INCREMENT statement without primary key option on the same column
190	MySQL doesn't support the COLUMN_VALUE pseudocolumn
191	MySQL doesn't support the OBJECT_ID pseudocolumn
192	MySQL doesn't support the ORA_ROWSCN pseudocolumn
193	MySQL doesn't support the ROWID pseudocolumn
198	MySQL doesn't support GLOBAL TEMPORARY TABLE
199	MySQL doesn't support CLUSTERED TABLE
200	MySQL doesn't support EXTERNAL TABLES
209	MySQL doesn't support virtual columns. It is emulated by triggers.
210	MySQL doesn't support FUNCTION or expressions AS DEFAULT VALUE. It is emulated by triggers.
215	MySQL doesn't support virtual columns with unsupported build-in functions
245	MySQL doesn't support views with nested table columns
296	Transformation from invalid table
348	MySQL doesn't support GLOBAL TEMPORARY TABLE
327	MySQL doesn't support the objects column

Constraints



Most constraints are automatically converted by AWS SCT as both Oracle and Aurora MySQL support entry-level ANSI compliance. These items include primary keys, foreign keys, null constraints, unique constraints, and default constraints (with some exceptions). Manual conversions are required for some foreign key cascading options. Check constraints are replaced with triggers by AWS SCT, and some default expressions for DateTime columns are not supported for automatic conversion. Complex expressions for other default values cannot be automatically converted by AWS SCT.

For more details, see [Constraints](#).

Action Code	Action Message
202	MySQL doesn't support foreign keys with different types of columns and referenced columns
203	Foreign keys with SET NULL action on column which is not nullable
204	BLOB and TEXT columns cannot be included in a foreign key
220	MySQL doesn't support the record type
325	MySQL does not support check constraints. Emulating triggers created
326	MySQL doesn't support constraints with the status DISABLED
591 / 593	This object uses a ROWID from the %s table

Data Types



Data type syntax and rules are similar between Oracle and Aurora MySQL and most are converted automatically by AWS SCT. Date and time handling paradigms are different for Oracle and Aurora MySQL and require manual verification or conversion. Also note that due to differences in data type behavior between Oracle and Aurora MySQL, manual verification and rigorous testing are highly recommended.

For more details, see [Data Types](#).

Action Code	Action Message
25	MySQL doesn't support assignment values to variables of INTERVAL datatype
28	Unable convert variable declaration of unsupported datatype
29	Unable convert variable usage of unsupported datatype
30	Unable convert variable usage of unsupported datatype
33	MySQL doesn't support fractional seconds for TIMESTAMP literals
212	MySQL doesn't have a data type BFILE

Common Table Expressions



Aurora MySQL version 5.7 does not support common table expressions; they cannot be automatically converted by AWS SCT.

For workarounds using traditional SQL syntax, see [Common Table Expressions](#).

Action Code	Action Message
127	MySQL doesn't support the recursive WITH clause

Cursors



The most commonly used cursor operations are converted automatically by AWS SCT. These operations include forward-only cursors, read-only cursors, DECLARE CURSOR, CLOSE CURSOR, and FETCH NEXT operations. Modifications through cursors and non-forward-only fetches, which are not supported by Aurora MySQL, require manual conversions.

For more details, see [Cursors](#).

Action Code	Action Message
31	Unable to convert CURSOR expression
84	MySQL doesn't support the cursor attribute SQL%ISOPEN
85	MySQL doesn't support the cursor attribute SQL%BULK_ROWCOUNT
297	MySQL doesn't support %ROWTYPE
330	MySQL doesn't support global cursors
337	MySQL doesn't support a variable of SYS_REFCURSOR type
343	Unable to transform the SELECT statement for a cursor
354	The dynamic SQL for the REF_CURSOR variable cannot be transformed
596	The conversion may be inaccurate
598	MySQL doesn't support the RETURN TYPE for the cursor

Transaction Isolation



Aurora MySQL supports the four transaction isolation levels specified in the SQL:92 standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE; all are automatically

converted by AWS SCT. AWS SCT also converts BEGIN / COMMIT and ROLLBACK commands that use slightly different syntax. Manual conversion is required for named, marked, and delayed durability transactions that are not supported by Aurora MySQL.

For more details, see [Transaction Isolation](#).

Action Code	Action Message
235	MySQL doesn't support PRAGMA options
302	MySQL doesn't support the NOWAIT clause in the LOCK TABLE statement
346	MySQL doesn't support the LOCK TABLE statement inside a stored procedure
350	The function cannot use statements that explicitly or implicitly begin or end a transaction, such as START TRANSACTION, COMMIT, or ROLLBACK

Stored Procedures



Aurora MySQL Stored Procedures provide very similar functionality to Oracle stored procedures and can be automatically converted by AWS SCT. Manual conversion is required for procedures that use RETURN values and some less common EXECUTE options such as the RECOMPILE and RESULTS SETS options.

For more details, see [Stored Procedures](#).

Action Code	Action Message
27	Unable convert object since there is no package body source code provided
152	The emulation might cover not all cases
234	MySQL doesn't support the EXCEPTION declaration
253	MySQL doesn't support the function %s with two parameters
266	MySQL doesn't support the function %s with the analytic clause
329	MySQL doesn't support the RAISE exception
331	MySQL doesn't support a global user exception
333	MySQL doesn't support EXCEPTION BLOCK in initialization blocks in packages
335	MySQL doesn't support the GOTO operator
340	MySQL doesn't support the %s function
342	MySQL doesn't support the %s exception
345	The handler might cover not all cases
350	The function cannot use statements that explicitly or implicitly begin or end a transaction, such as START TRANSACTION, COMMIT, or ROLLBACK
590	The function was converted as procedure

Triggers



Aurora MySQL supports BEFORE and AFTER triggers for INSERT, UPDATE, and DELETE. However, Aurora MySQL triggers differ substantially from Oracle triggers, but most common use cases can be migrated with minimal code changes. Although AWS SCT can automatically migrate trigger code, manual inspection and potential code modifications may be required because Aurora MySQL triggers are executed once per row, not once per statement as with Oracle triggers.

For more details, see [Triggers](#).

Action Code	Action Message
236	MySQL doesn't support INSTEAD OF triggers
237	MySQL doesn't support statement triggers
238	MySQL doesn't support the REFERENCING clauses
239	MySQL doesn't support WHEN(condition) triggers
240	MySQL doesn't support triggers on nested table columns in views
241	MySQL doesn't support the FOLLOWS PRECEDES clause
242	MySQL doesn't support COMPOUND TRIGGER
243	MySQL doesn't support the UPDATE OF clause
244	MySQL doesn't support conditional predicates
306	Transformation from invalid trigger
310	MySQL doesn't support triggers for views
311	MySQL doesn't support system triggers
312	MySQL doesn't support the DISABLED clause
313	MySQL doesn't support the action-type clause
314	MySQL doesn't support crossedition triggers
316	MySQL doesn't support the APPLY-SERVER-ONLY clause
317	MySQL doesn't support the PARENT referencing clause
415	MySQL doesn't support system triggers
524	MySQL doesn't support triggers for multiple events
588	MySQL doesn't support more than one trigger per event. A few triggers were merged

Sequences



Although the syntax for Oracle IDENTITY and Aurora MySQL AUTO_INCREMENT auto-enumeration columns differs significantly, it can be automatically converted by AWS SCT. Some limitations imposed by Aurora MySQL

require manual conversions such as explicit SEED and INCREMENT auto-enumeration columns that are not part of the primary key and the table-independent SEQUENCE objects.

For more details, see [Sequences and Identity](#).

Action Code	Action Message
341	MySQL doesn't support sequences

Date and Time Functions



The most commonly used date and time functions are automatically converted by AWS SCT despite the significant difference in syntax. Be aware of differences in data types, time zone awareness, and locale handling as well the functions themselves and inspect the expression value output carefully. Some less commonly used options such as millisecond, nanosecond, and time zone offsets require manual conversion.

For more details, see [Date and Time Functions](#).

Action Code	Action Message
213	MySQL expands fractional seconds support for TIME, DATETIME, and TIMESTAMP values, with up to microseconds (6 digits) of precision
214	MySQL doesn't have a data type that stores time zone information. The DATETIME data type stores timestamps in the MyOracle time zone
216	MySQL expands fractional seconds support for TIME, DATETIME, and TIMESTAMP values, with up to microseconds (6 digits) of precision. MySQL doesn't have a data type that stores time zone information

User Defined Types



Aurora MySQL 5.7 does not support user-defined types or user-defined table-valued parameters. AWS SCT can convert standard user-defined types by replacing them with their base types, but manual conversion is required for user-defined table types, which are used for table-valued parameters for stored procedures.

For more details, see [User Defined Types](#).

Action Code	Action Message
196	MySQL doesn't support OBJECT TABLE
218	MySQL doesn't support user types

Synonyms



Aurora MySQL version 5.7 does not support synonyms; they cannot be automatically converted by AWS SCT.

Action Code	Action Message
352	MySQL doesn't support synonyms

XML



Aurora MySQL provides minimal support for XML, but it does offer a native JSON data type and more than 25 dedicated JSON functions. Despite these differences, the most commonly used basic XML functions can be automatically migrated by AWS SCT. Some options, used in functions or with sub-queries, such as EXPLICIT, require manual conversion.

For more details, see [XML](#).

Action Code	Action Message
194	MySQL doesn't support XMLTYPE tables
195	MySQL doesn't support the XMLDATA pseudocolumn
303	MySQL doesn't support the XMLTable function

MERGE



Aurora MySQL version 5.7 does not support the MERGE statement; it cannot be automatically converted by AWS SCT. Manual conversion is straightforward in most cases.

Action Code	Action Message
102	MySQL doesn't support the MERGE statement

Query hints



Basic query hints such as index hints, can be converted automatically by AWS SCT, except for DML statements. Note that specific optimizations used for Oracle may not be applicable to the MySQL query optimizer. We recommend that you start migration testing with all hints removed. Then, selectively apply hints as a last resort if other means such as schema, index, and query optimizations have failed. Plan guides are not supported by Aurora MySQL.

For more details, see [Query hints and Plan Guides](#).

Action Code	Action Message
103	MySQL doesn't support the %s hint

Indexes



Basic non-clustered indexes, which are the most commonly used type of indexes, are automatically migrated by AWS SCT. User-defined clustered indexes are not supported by Aurora MySQL as they are always created for the primary key. In addition, filtered indexes, indexes with included columns, and some Oracle specific index options cannot be migrated automatically and require manual conversion.

For more details, see [Indexes](#).

Action Code	Action Message
205	MySQL has the InnoDB internal maximum key length
206	MySQL doesn't support bitmap indexes
207	MySQL doesn't support function indexes
208	MySQL doesn't support domain indexes
328	Unable to transform index in INVALID state

Partitioning



Because of how Aurora MySQL stores each table in its own file, and since file management is performed by AWS and cannot be modified, some of the physical aspects of partitioning in Oracle do not apply to Aurora MySQL. Due to the vast differences between partition creation, query, and management between Aurora MySQL and Oracle, AWS SCT does not automatically convert table and index partitions. These items require manual conversion.

For more details, see [Partitioning](#).

Action Code	Action Message
201	MySQL doesn't support all partition types
699	MySQL doesn't support not allowed partitions functions

Materialized View



Aurora MySQL version 5.7 does not support Materialized Views; they cannot be automatically converted by AWS SCT.

For more details, see [Materialized Views](#).

Action Code	Action Message
94 / 95	MySQL doesn't support a materialized VIEW

Views



Although the basic syntax for creating a view in Oracle and Aurora MySQL is almost identical, there are some sub-options that can differ significantly, requiring additional manual migration tasks.

For more details, see [Views](#).

Action Code	Action Message
75	MySQL doesn't support VIEW with the READ ONLY option
93	You cannot perform this action by using the UPDATE statement on a VIEW
97	You cannot perform this action by using the DELETE statement on a VIEW
320	Transformation from invalid view
321	MySQL doesn't support the object view
323	MySQL doesn't support the subview from superview
324	MySQL doesn't support the editioning view
583	MySQL doesn't support constraints for view

UTL_Mail and UTL_SMTP



Aurora MySQL does not provide native support for sending email from the database.

For more details and potential workarounds, see [Database Mail](#).

Action Code	Action Message
81	MySQL doesn't support sending notifications by using SMS
82	MySQL doesn't support sending e-mail

Database Links



Aurora MySQL does not support remote data access. Connectivity between schemas is trivial, but connectivity to other instances requires a custom solution; it can not be automatically converted by AWS SCT.

For more details, see [Database Links](#).

Action Code	Action Message
600	MySQL doesn't support use of dblink

PLSQL



AWS SCT automatically converts the most commonly used SQL statements as both Oracle and Aurora MySQL support the entry level ANSI compliance. Some changes may be required for DML related to ERROR LOG, subquery, and partitions.

Action Code	Action Message
63	Unable to convert the UPDATE statement with multiple-column subquery in SET clause
64	MySQL doesn't support the UPDATE statement with the ERROR LOG option
65	MySQL doesn't support the UPDATE statement for a subquery
66	MySQL doesn't support the UPDATE statement for the RETURNING option
67	MySQL doesn't support the DELETE statement with the ERROR LOG option
68	MySQL doesn't support the DELETE statement for a subquery
69	MySQL doesn't support the DELETE statement with the RETURNING option
70	MySQL doesn't support the INSERT statement with the ERROR LOG option
71	MySQL doesn't support the INSERT statement for a subquery
72	MySQL doesn't support the INSERT statement with the RETURNING option
77	MySQL doesn't support the PIVOT clause for the SELECT statement
78	MySQL doesn't support the UNPIVOT clause for the SELECT statement
87	MySQL doesn't support the RETURNING BULK COLLECT INTO clause
89	MySQL doesn't support the INSERT statement for a VIEW
90	MySQL doesn't support the INSERT statement for a SUBPARTITION
122	MySQL doesn't support hierarchical queries
125	MySQL doesn't support the GROUPING SETS statement
128	MySQL doesn't support ORACLE FLASHBACK VERSION QUERY

Action Code	Action Message
138	MySQL doesn't support FOR UPDATE OF
139	MySQL doesn't support FOR UPDATE SKIP LOCKED
140	MySQL doesn't support BULK COLLECT INTO
141	MySQL doesn't support ORDER BY ... NULLS FIRST
143	MySQL doesn't support FOR UPDATE NOWAIT clauses
144	MySQL doesn't support FOR UPDATE WAIT clauses
585	Unable to convert outer join inside a correlated query
594	MySQL doesn't support LATERAL neither CROSS APPLY nor OUTER APPLY correlated inline views
599	MySQL doesn't support use of a CURRENT OF clause for data manipulation language (DML) queries that are in the body of a cursor loop. A manual conversion is required

EXECUTE IMMEDIATE



There is a major difference between Oracle and Aurora MySQL for the EXECUTE IMMEDIATE statement. In MySQL, this statement must be used after a PREPARE command. Executing SQL with results and bind variables, and executing Anonymous blocks, are not supported.

For more details, see [Execute Immediate](#).

Action Code	Action Message
88	MySQL doesn't support the EXECUTE IMMEDIATE statement with BULK COLLECT
334	MySQL doesn't support the dynamic SQL statement EXECUTE IMMEDIATE
336	MySQL doesn't support the dynamic SQL statement EXECUTE IMMEDIATE with %s clause

DBMS_OUTPUT



Aurora MySQL does not provide native support for the dbms_output procedure. Use the RAISE command instead.

For more details, see [DBMS_OUTPUT](#).

Action Code	Action Message
332	MySQL doesn't support the procedure dbms_output.put_line
349	MySQL doesn't support the procedure dbms_output.put

AWS Database Migration Service (DMS)

Usage

The AWS Database Migration Service (DMS) helps you migrate databases to AWS quickly and securely. The source database remains fully operational during the migration, minimizing downtime to applications that rely on the database. The AWS Database Migration Service can migrate your data to and from most widely-used commercial and open-source databases.

The service supports homogenous migrations such as Oracle to Oracle as well as heterogeneous migrations between different database platforms such as Oracle to Amazon Aurora or Microsoft SQL Server to MySQL. It also allows you to stream data to Amazon Redshift, Amazon DynamoDB, and Amazon S3 from any of the supported sources, which are Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, SAP ASE, SQL Server, IBM DB2 LUW, and MongoDB, enabling consolidation and easy analysis of data in a petabyte-scale data warehouse. The AWS Database Migration Service can also be used for continuous data replication with high-availability.

When migrating databases to Aurora, Redshift or DynamoDB, you can [use DMS free for six months](#).

For all supported sources for DMS, see

https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.html

For all supported targets for DMS, see

https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Target.html

Migration Tasks Performed by AWS DMS

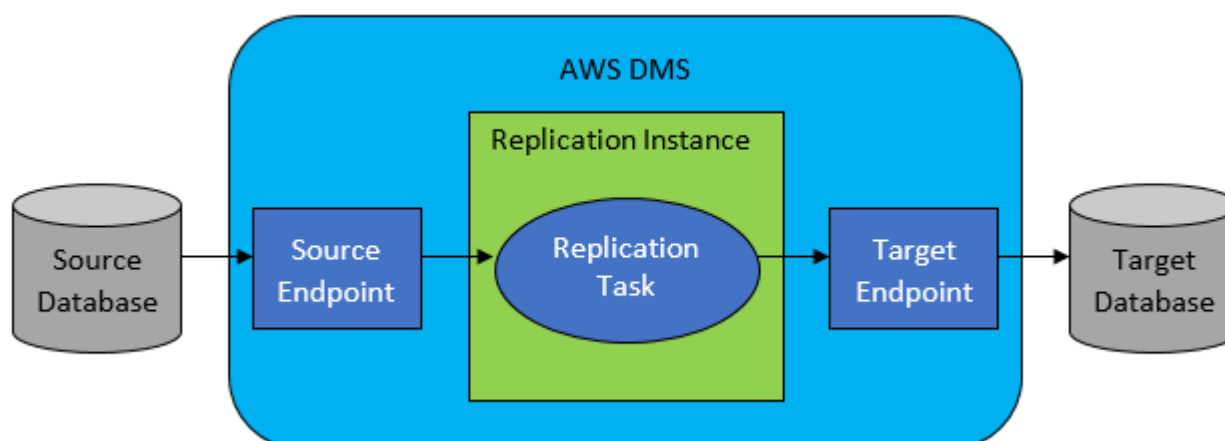
- In a traditional solution, you need to perform capacity analysis, procure hardware and software, install and administer systems, and test and debug the installation. AWS DMS automatically manages the deployment, management, and monitoring of all hardware and software needed for your migration. Your migration can be up and running within minutes of starting the AWS DMS configuration process.
- With AWS DMS, you can scale up (or scale down) your migration resources as needed to match your actual workload. For example, if you determine that you need additional storage, you can easily increase your allocated storage and restart your migration, usually within minutes. On the other hand, if you discover that you aren't using all of the resource capacity you configured, you can easily downsize to meet your actual workload.
- AWS DMS uses a pay-as-you-go model. You only pay for AWS DMS resources while you use them as opposed to traditional licensing models with up-front purchase costs and ongoing maintenance charges.
- AWS DMS automatically manages all of the infrastructure that supports your migration server including hardware and software, software patching, and error reporting.
- AWS DMS provides automatic failover. If your primary replication server fails for any reason, a backup replication server can take over with little or no interruption of service.
- AWS DMS can help you switch to a modern, perhaps more cost-effective database engine than the one you are running now. For example, AWS DMS can help you take advantage of the managed database services provided by Amazon RDS or Amazon Aurora. Or, it can help you move to the managed data warehouse service provided by Amazon Redshift, NoSQL platforms like Amazon DynamoDB, or low-cost storage platforms like Amazon S3. Conversely, if you want to migrate away from old infrastructure but continue to use the same database engine, AWS DMS also supports that process.

- AWS DMS supports nearly all of today's most popular DBMS engines as data sources, including Oracle, Microsoft SQL Server, MySQL, MariaDB, PostgreSQL, Db2 LUW, SAP, MongoDB, and Amazon Aurora.
- AWS DMS provides a broad coverage of available target engines including Oracle, Microsoft SQL Server, PostgreSQL, MySQL, Amazon Redshift, SAP ASE, S3, and Amazon DynamoDB.
- You can migrate from any of the supported data sources to any of the supported data targets. AWS DMS supports fully heterogeneous data migrations between the supported engines.
- AWS DMS ensures that your data migration is secure. Data at rest is encrypted with AWS Key Management Service (AWS KMS) encryption. During migration, you can use Secure Socket Layers (SSL) to encrypt your in-flight data as it travels from source to target.

How AWS DMS Works

At its most basic level, AWS DMS is a server in the AWS Cloud that runs replication software. You create a source and target connection to tell AWS DMS where to extract from and load to. Then, you schedule a task that runs on this server to move your data. AWS DMS creates the tables and associated primary keys if they don't exist on the target. You can pre-create the target tables manually if you prefer. Or you can use AWS SCT to create some or all of the target tables, indexes, views, triggers, and so on.

The following diagram illustrates the AWS DMS process.



Latest updates

DMS is continuously evolving and supporting more and more options, find below some of the updates add since last edition of this playbook:

- CDC tasks and Oracle source tables created using CREATE TABLE AS AWS -DMS now supports both full-load and CDC and CDC-only tasks running against Oracle source tables created using the CREATE TABLE AS statement.

- New MySQL version AWS DMS now supports MySQL version 8.0 as a source except when the transaction payload is compressed.
- Support for AWS Secrets Manager integration You can store the database connection details (user credentials) for supported endpoints securely in AWS Secrets Manager. You can then submit the corresponding secret instead of plain-text credentials to AWS DMS when you create or modify an endpoint. AWS DMS then connects to the endpoint databases using the secret. For more information on creating secrets for AWS DMS endpoints see [Using secrets to access AWS Database Migration Service endpoints](#).
- Support for Oracle extended data types Oracle extended data types for both Oracle source and targets are now supported.
- TLS 1.2 support for MySQL AWS DMS now supports TLS 1.2 for MySQL endpoints.
- TLS 1.2 support for SQL Server AWS DMS now supports TLS 1.2 for SQL Server endpoints.

For a complete guide with a step-by-step walkthrough including all the latest notes for migrating SQL Server to Aurora MySQL (which is very similar to the Oracle-PostgreSQL migration process) with DMS, see https://docs.aws.amazon.com/dms/latest/sbs/CHAP_SQLServer2Aurora.html

For more information about DMS, see:

- <https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html>
- https://docs.aws.amazon.com/dms/latest/userguide/CHAP_BestPractices.html

Amazon RDS on Outposts

PLEASE NOTE, ENTIRE TOPIC IS RELATED TO RDS AND IS NOT SUPPORTED WITH AURORA

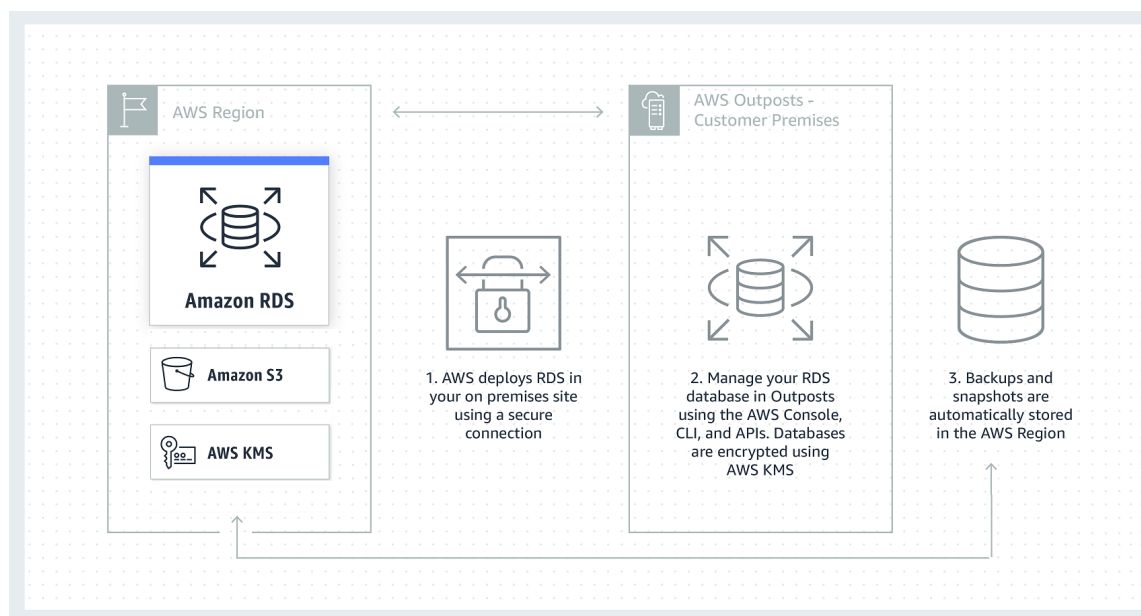
Usage

Amazon RDS on Outposts is a fully managed service that offers the same AWS infrastructure, AWS services, APIs, and tools to virtually any data center, co-location space, or on-premises facility for a truly consistent hybrid experience. Amazon RDS on Outposts is ideal for workloads that require low latency access to on-premises systems, local data processing, data residency, and migration of applications with local system inter-dependencies.

When you deploy Amazon RDS on Outposts, you can run RDS on premises for low latency workloads that need to be run in close proximity to your on-premises data and applications. Amazon RDS on Outposts also enables automatic backup to an AWS Region. You can manage RDS databases both in the cloud and on premises using the same AWS Management Console, APIs, and CLI. Amazon RDS on Outposts supports Microsoft SQL Server, MySQL, and PostgreSQL database engines, with support for additional database engines coming soon.

How it works

Amazon RDS on Outposts lets you run Amazon RDS in your on-premises or co-location site. You can deploy and scale an RDS database instance in Outposts just as you do in the cloud, using the AWS console, APIs, or CLI. RDS databases in Outposts are encrypted at rest using AWS KMS keys. RDS automatically stores all automatic backups and manual snapshots in the AWS Region.



This option is helpful when you need to run RDS on premises for low latency workloads that need to be run in close proximity to your on-premises data and applications

For more information, see:

- <https://aws.amazon.com/outposts/>
- <https://aws.amazon.com/rds/outposts/>
- <https://aws.amazon.com/blogs/aws/new-create-amazon-rds-db-instances-on-aws-outposts/>

Amazon RDS Proxy

Amazon RDS Proxy is a fully managed, highly available database proxy for Amazon Relational Database Service (RDS) that makes applications more scalable, more resilient to database failures, and more secure.

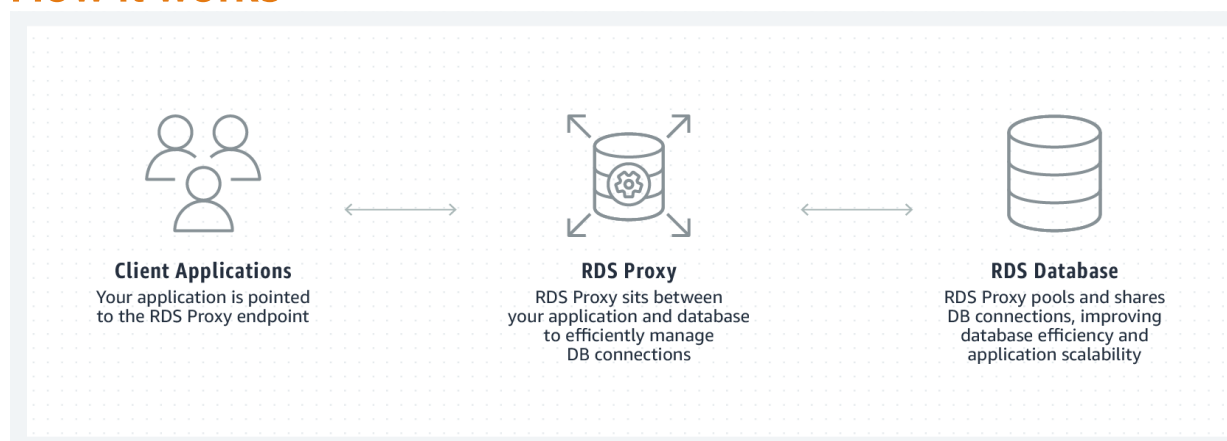
Many applications, including those built on modern server-less architectures, can have many open connections to the database server, and may open and close database connections at a high rate, exhausting database memory and compute resources. Amazon RDS Proxy allows applications to pool and share connections established with the database, improving database efficiency and application scalability. With RDS Proxy, fail-over times for Aurora and RDS databases are reduced by up to 66% and database credentials, authentication, and access can be managed through integration with AWS Secrets Manager and AWS Identity and Access Management (IAM).

Amazon RDS Proxy can be enabled for most applications with no code changes, and you don't need to provision or manage any additional infrastructure. Pricing is simple and predictable: you pay per vCPU of the database instance for which the proxy is enabled. Amazon RDS Proxy is now generally available for Aurora MySQL, Aurora PostgreSQL, RDS MySQL and RDS PostgreSQL.

Benefits

- **Improved application performance**
Amazon RDS proxy manages a connection pooling which helps with reducing the stress on database compute and memory resources that typically occurs when new connections are established and it is useful to efficiently support a large number and frequency of application connections
- **Increase application availability**
By automatically connecting to a new database instance while preserving application connections Amazon RDS Proxy can reduce fail-over time by 66%
- **Manage application security**
RDS Proxy also enables you to centrally manage database credentials using AWS Secrets Manager
- **Fully managed**
Amazon RDS Proxy gives you the benefits of a database proxy without requiring additional burden of patching and managing your own proxy server.
- **Fully compatible with your database**
Amazon RDS Proxy is fully compatible with the protocols of supported database engines, so you can deploy RDS Proxy for your application without making changes to your application code.
- **Available and durable**
Amazon RDS Proxy is highly available and deployed over multiple Availability Zones (AZs) to protect you from infrastructure failure

How it works



For more information, see:

- <https://aws.amazon.com/blogs/aws/amazon-rds-proxy-now-generally-available/>
- <https://aws.amazon.com/rds/proxy/>

Amazon Aurora Serverless v1

Usage

Amazon Aurora Serverless v1 (Amazon Aurora Serverless version 1) is an on-demand autoscaling configuration for Amazon Aurora. An Aurora Serverless DB cluster is a DB cluster that scales compute capacity up and down based on your application's needs. This contrasts with Aurora provisioned DB clusters, for which you manually manage capacity. Aurora Serverless v1 provides a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads. It is cost-effective because it automatically starts up, scales compute capacity to match your application's usage, and shuts down when it's not in use.

To learn more about pricing, see Serverless Pricing under MySQL-Compatible Edition or PostgreSQL-Compatible Edition on the Amazon Aurora pricing page.

Aurora Serverless v1 clusters have the same kind of high-capacity, distributed, and highly available storage volume that is used by provisioned DB clusters. The cluster volume for an Aurora Serverless v1 cluster is always encrypted. You can choose the encryption key, but you can't disable encryption. That means that you can perform the same operations on an Aurora Serverless v1 that you can on encrypted snapshots. For more information, see Aurora Serverless v1 and snapshots.

Aurora Serverless v1 provides the following advantages:

- **Simpler than provisioned** - Aurora Serverless v1 removes much of the complexity of managing DB instances and capacity.
- **Scalable** - Aurora Serverless v1 seamlessly scales compute and memory capacity as needed, with no disruption to client connections.
- **Cost-effective** - When you use Aurora Serverless v1, you pay only for the database resources that you consume, on a per-second basis.
- **Highly available storage** - Aurora Serverless v1 uses the same fault-tolerant, distributed storage system with six-way replication as Aurora to protect against data loss.

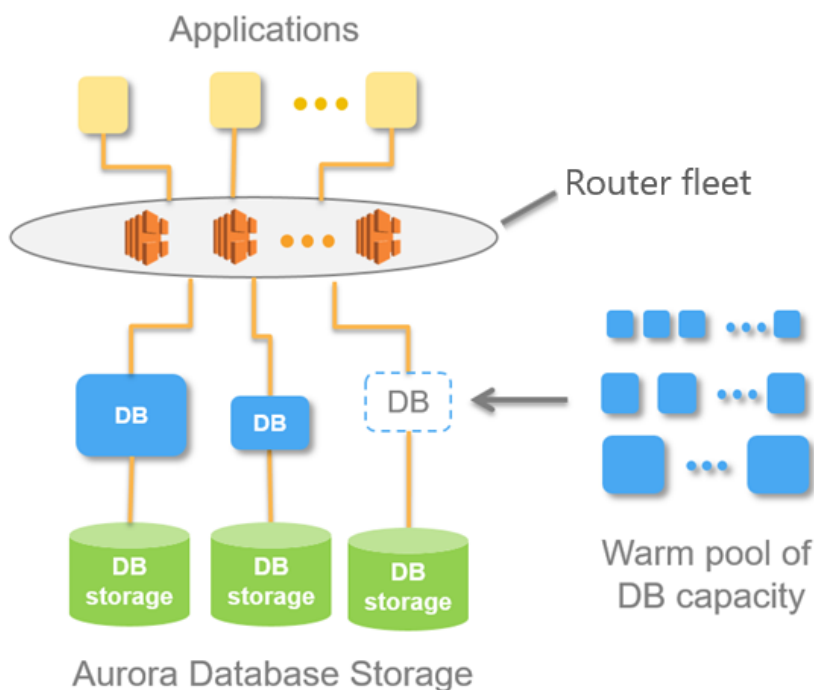
Aurora Serverless v1 is designed for the following use cases:

- **Infrequently used applications** - You have an application that is only used for a few minutes several times per day or week, such as a low-volume blog site. With Aurora Serverless v1, you pay for only the database resources that you consume on a per-second basis.
- **New applications** - You're deploying a new application and you're unsure about the instance size you need. By using Aurora Serverless v1, you can create a database endpoint and have the database auto-scale to the capacity requirements of your application.
- **Variable workloads** - You're running a lightly used application, with peaks of 30 minutes to several hours a few times each day, or several times per year. Examples are applications for human resources, budgeting, and operational reporting applications. With Aurora Serverless v1, you no longer need to provision for peak

or average capacity.

- **Unpredictable workloads** - You're running daily workloads that have sudden and unpredictable increases in activity. An example is a traffic site that sees a surge of activity when it starts raining. With Aurora Serverless v1, your database autoscales capacity to meet the needs of the application's peak load and scales back down when the surge of activity is over.
- **Development and test databases** - Your developers use databases during work hours but don't need them on nights or weekends. With Aurora Serverless v1, your database automatically shuts down when it's not in use.
- **Multi-tenant applications** - With Aurora Serverless v1, you don't have to individually manage database capacity for each application in your fleet. Aurora Serverless v1 manages individual database capacity for you.

This process takes almost no time and since the storage is shared between nodes Aurora can scale up or down in seconds for most workloads. The service currently has autoscaling thresholds of 1.5 minutes to scale up and 5 minutes to scale down. That means metrics must exceed the limits for 1.5 minutes to trigger a scale up or fall below the limits for 5 minutes to trigger a scale down. The cool-down period between scaling activities is 5 minutes to scale up and 15 minutes to scale down. Before scaling can happen the service has to find a “scaling point” which may take longer than anticipated if you have long-running transactions. Scaling operations are transparent to the connected clients and applications since existing connections and session state are transferred to the new nodes. The only difference with pausing and resuming is a higher latency for the first connection, typically around 25 seconds. You can find more details in the documentation.



Aurora Serverless v2 (preview)

Note: Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

Amazon Aurora Serverless v2 (preview) has been architected from the ground up to support serverless DB clusters that are instantly scalable. The Aurora Serverless v2 (preview) architecture rests on a lightweight foundation that's engineered to provide the security and isolation needed in multitenant serverless cloud environments. This foundation has very little overhead so it can respond quickly. It's also powerful enough to meet dramatic increases in processing demand.

When you create your Aurora Serverless v2 (preview) DB cluster, you define its capacity as a range between minimum and maximum number of Aurora capacity units (ACUs):

- Minimum Aurora capacity units - The smallest number of ACUs down to which your Aurora Serverless v2 (preview) DB cluster can scale.
- Maximum Aurora capacity units - The largest number of ACUs up to which your Aurora Serverless v2 (preview) DB cluster can scale.

Each ACU provides 2 GiB (gibibytes) of memory (RAM) and associated virtual processor (vCPU) with networking.

Unlike Aurora Serverless v1, which scales by doubling ACUs each time the DB cluster reaches a threshold, Aurora Serverless v2 (preview) can increase ACUs incrementally. When your workload demand begins to reach the current resource capacity, your Aurora Serverless v2 (preview) DB cluster scales the number of ACUs. Your cluster scales ACUs in the increments required to provide the best performance for the resources consumed.

How to provision

Browse to the [Databases page](#) and click on "Create database"

Pick "Serverless" capacity type

Engine options

Engine type [Info](#)

Amazon Aurora

MySQL

MariaDB

PostgreSQL

Oracle

Microsoft SQL Server

Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

Capacity type [Info](#)

Provisioned
You provision and manage the server instance sizes.

Serverless
You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

Version

Aurora (MySQL 5.7) 2.07.1 ▼

To see more versions, modify the capacity types. [Info](#)

Aurora MySQL engine versions earlier than 2.09.1 don't support the newest r6g generation instance classes.

Choose the capacity properties suite for you use case

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit [Info](#)

1
2GB RAM
▼

Maximum Aurora capacity unit [Info](#)

64
122GB RAM
▼

▼ **Additional scaling configuration**

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

For more information, see:

- <https://aws.amazon.com/rds/aurora/serverless/>
- <https://aws.amazon.com/blogs/aws/aurora-serverless-ga/>
- <https://aws.amazon.com/blogs/aws/amazon-aurora-postgresql-serverless-now-generally-available/>

Amazon Aurora for MySQL Parallel Query

Usage

Amazon Aurora Parallel Query is a feature of the Amazon Aurora database that provides faster analytical queries over your current data, without having to copy the data into a separate system. It can speed up queries by up to two orders of magnitude, while maintaining high throughput for your core transactional workload.

While some databases can parallelize query processing across CPUs in one or a handful of servers, Parallel Query takes advantage of Aurora's unique architecture to push down and parallelize query processing across thousands of CPUs in the Aurora storage layer. By offloading analytical query processing to the Aurora storage layer, Parallel Query reduces network, CPU, and buffer pool contention with the transactional workload.

Features

Accelerate Your Analytical Queries

In a traditional database, running analytical queries directly on the database means accepting slower query performance and risking a slowdown of your transactional workload, even when running light queries. Queries can run for several minutes to hours, depending on the size of the tables and database server instances. Queries are also slowed down by network latency, since the storage layer may have to transfer entire tables to the database server for processing.

With Parallel Query, query processing is pushed down to the Aurora storage layer. The query gains a large amount of computing power, and it needs to transfer far less data over the network. In the meantime, the Aurora database instance can continue serving transactions with much less interruption. This way, you can run transactional and analytical workloads alongside each other in the same Aurora database, while maintaining high performance.

Query on Fresh Data

Many analytical workloads require both fresh data and good query performance. For example, operational systems such as network monitoring, cyber-security or fraud detection rely on fresh, real-time data from a transactional database, and can't wait for it to be extracted to a analytics system.

By running your queries in the same database that you use for transaction processing, without degrading transaction performance, Amazon Aurora Parallel Query enables smarter operational decisions with no additional software and no changes to your queries.

Benefits of using Parallel query

- Improved I/O performance, due to parallelizing physical read requests across multiple storage nodes.
- Reduced network traffic. Aurora doesn't transmit entire data pages from storage nodes to the head node and then filter out unnecessary rows and columns afterward. Instead, Aurora transmits compact tuples containing only the column values needed for the result set.
- Reduced CPU usage on the head node, due to pushing down function processing, row filtering, and column projection for the WHERE clause.

- Reduced memory pressure on the buffer pool. The pages processed by the parallel query aren't added to the buffer pool. This approach reduces the chance of a data-intensive scan evicting frequently used data from the buffer pool.
- Potentially reduced data duplication in your extract, transform, load (ETL) pipeline, by making it practical to perform long-running analytic queries on existing data.

Important notes

- Table Formats - The table row format must be COMPACT; partitioned tables are not supported.
- Data Types - The TEXT, BLOB, and GEOMETRY data types are not supported.
- DDL - The table cannot have any pending fast online DDL operations.
- Cost - You can make use of Parallel Query at no extra charge. However, because it makes direct access to storage, there is a possibility that your IO cost will increase.

For more information, see: <https://aws.amazon.com/rds/aurora/parallel-query/>

Amazon Aurora Backtrack

Usage

We've all been there, you need to make a quick, seemingly simple fix to an important production database. You compose the query, give it a once-over, and let it run. Seconds later you realize that you forgot the WHERE clause, dropped the wrong table, or made another serious mistake, and interrupt the query, but the damage has been done. You take a deep breath, whistle through your teeth, wish that reality came with an Undo option.

Backtracking "rewinds" the DB cluster to the time you specify. Backtracking is not a replacement for backing up your DB cluster so that you can restore it to a point in time. However, backtracking provides the following advantages over traditional backup and restore:

- You can easily undo mistakes. If you mistakenly perform a destructive action, such as a DELETE without a WHERE clause, you can backtrack the DB cluster to a time before the destructive action with minimal interruption of service.
- You can backtrack a DB cluster quickly. Restoring a DB cluster to a point in time launches a new DB cluster and restores it from backup data or a DB cluster snapshot, which can take hours. Backtracking a DB cluster doesn't require a new DB cluster and rewinds the DB cluster in minutes.
- You can explore earlier data changes. You can repeatedly backtrack a DB cluster back and forth in time to help determine when a particular data change occurred. For example, you can backtrack a DB cluster three hours and then backtrack forward in time one hour. In this case, the backtrack time is two hours before the original time.

Aurora uses a distributed, log-structured storage system (read Design Considerations for High Throughput Cloud-Native Relational Databases to learn a lot more); each change to your database generates a new log record, identified by a Log Sequence Number (LSN). Enabling the backtrack feature provisions a FIFO buffer in the cluster for storage of LSNs. This allows for quick access and recovery times measured in seconds.

When you create a new Aurora MySQL DB cluster, backtracking is configured when you choose Enable Backtrack and specify a Target Backtrack window value that is greater than zero in the Backtrack section.

To create a DB cluster, follow the instructions in [Creating an Amazon Aurora DB cluster](#). The following image shows the Backtrack section.

Backtrack

Backtrack lets you quickly rewind the DB cluster to a specific point in time, without having to create another DB cluster. [Info](#)

Enable Backtrack

Enabling Backtrack will charge you for storing the changes you make for backtracking.

Target Backtrack window

The Backtrack window determines how far back in time you could go. Aurora will try to retain enough log information to support that window of time. [Info](#)

hours (up to 72)

Typical user cost

The cost of Backtrack depends on how often you are updating your database. This is an estimate based on typical workloads for your selected instance size (db.r5.large). [Info](#)

\$ 14.38 USD / month

After a production error, you can simply pause your application, open up the Aurora Console, select the cluster, and click Backtrack DB cluster

Then you select Backtrack and choose the point in time just before your epic fail, and click Backtrack DB cluster:

Backtrack DB cluster

Rewinds the DB cluster to a previous point in time without creating a new DB cluster.

Earliest restorable time is June 16, 2021 at 8:53:02 PM UTC-4 (Local) ⓘ

Date: Time: : : UTC-4

The next available time will be used if the specified time is not available.

⚠ Your DB cluster is unavailable during the Backtrack process, which typically takes a few minutes.

Cancel Backtrack DB cluster

Then you wait for the rewind to take place, unpause your application and proceed as if nothing had happened. When you initiate a backtrack, Aurora will pause the database, close any open connections, drop uncommitted writes, and wait for the backtrack to complete. Then it will resume normal operation and be able to accept requests. The instance state will be backtracking while the rewind is underway.

Backtrack window

With backtracking, there is a target backtrack window and an actual backtrack window:

- The target backtrack window is the amount of time you want to be able to backtrack your DB cluster. When you enable backtracking, you specify a target backtrack window. For example, you might specify a target backtrack window of 24 hours if you want to be able to backtrack the DB cluster one day.
- The actual backtrack window is the actual amount of time you can backtrack your DB cluster, which can be smaller than the target backtrack window. The actual backtrack window is based on your workload and the storage available for storing information about database changes, called change records.

As you make updates to your Aurora DB cluster with backtracking enabled, you generate change records. Aurora retains change records for the target backtrack window, and you pay an hourly rate for storing them. Both the target backtrack window and the workload on your DB cluster determine the number of change records you store. The workload is the number of changes you make to your DB cluster in a given amount of time. If your workload is heavy, you store more change records in your backtrack window than you do if your workload is light.

You can think of your target backtrack window as the goal for the maximum amount of time you want to be able to backtrack your DB cluster. In most cases, you can backtrack the maximum amount of time that you specified. However, in some cases, the DB cluster can't store enough change records to backtrack the maximum amount of time, and your actual backtrack window is smaller than your target. Typically, the actual backtrack window is smaller than the target when you have extremely heavy workload on your DB cluster. When your actual backtrack window is smaller than your target, we send you a notification.

When backtracking is enabled for a DB cluster, and you delete a table stored in the DB cluster, Aurora keeps that table in the backtrack change records. It does this so that you can revert back to a time before you deleted the table. If you don't have enough space in your backtrack window to store the table, the table might be removed from the backtrack change records eventually.

Backtracking limitations

The following limitations apply to backtracking:

- Backtracking an Aurora DB cluster is available in certain AWS Regions and for specific Aurora MySQL versions only. For more information, see [Backtracking in Aurora](#).
- Backtracking is only available for DB clusters that were created with the Backtrack feature enabled. You can enable the Backtrack feature when you create a new DB cluster or restore a snapshot of a DB cluster. For DB clusters that were created with the Backtrack feature enabled, you can create a clone DB cluster with the Backtrack feature enabled. Currently, you can't perform backtracking on DB clusters that were created with the Backtrack feature disabled.
- The limit for a backtrack window is 72 hours.
- Backtracking affects the entire DB cluster. For example, you can't selectively backtrack a single table or a single data update.
- Backtracking isn't supported with binary log (binlog) replication. Cross-Region replication must be disabled before you can configure or use backtracking.
- You can't backtrack a database clone to a time before that database clone was created. However, you can use the original database to backtrack to a time before the clone was created. For more information about database cloning, see [Cloning an Aurora DB cluster volume](#).

- Backtracking causes a brief DB instance disruption. You must stop or pause your applications before starting a backtrack operation to ensure that there are no new read or write requests. During the backtrack operation, Aurora pauses the database, closes any open connections, and drops any uncommitted reads and writes. It then waits for the backtrack operation to complete.
- Backtracking isn't supported for the following AWS Regions:
 - Africa (Cape Town)
 - China (Ningxia)
 - Asia Pacific (Hong Kong)
 - Europe (Milan)
 - Europe (Stockholm)
 - Middle East (Bahrain)
 - South America (São Paulo)
- You can't restore a cross-Region snapshot of a backtrack-enabled cluster in an AWS Region that doesn't support backtracking.
- You can't use Backtrack with Aurora multi-master clusters.
- If you perform an in-place upgrade for a backtrack-enabled cluster from Aurora MySQL version 1 to version 2, you can't backtrack to a point in time before the upgrade happened

For more information, see: <https://aws.amazon.com/blogs/aws/amazon-aurora-backtrack-turn-back-time/>

Migration Quick Tips

This section provides migration tips that can help save time as you transition from Oracle to Aurora MySQL. It addresses many of the challenges faced by administrators new to Aurora MySQL. Some of these tips describe functional differences in similar features between Oracle and Aurora MySQL.

Management

- In Aurora MySQL, the term "Database Snapshot" is equivalent to Oracle's RMAN backup.
- Partitioning in Aurora MySQL does not provide many of Oracle's management features such as Partition Advisor, Preference Partitioning, Virtual Column-Based Partitioning, and Automatic List Partitioning.
- Unlike Oracle's statistics, Aurora MySQL does not collect detailed key/value distributions in tables; it only collects statistics on indexes.
- Amazon services, such as Lambda, can be used to replicate functionality of features not provided by MySQL (for example, email).
- Amazon RDS manages parameters and backups. It is very useful for checking a parameter's value against its default and comparing them to another parameter group.
- With just a few clicks, you can create Replicas to implement High Availability.
- Aurora MySQL does not have an equivalent to database links. It can only query across databases within the same instance.



SQL

- Aurora MySQL does not support statement-level triggers or triggers on system events.
- Aurora MySQL does not support many cursor status checks. When declaring cursors in Aurora MySQL, you must create an explicit HANDLER object.
- To execute a stored procedure (function), use CALL instead of EXECUTE.
- To execute a string as a query, use Aurora MySQL Prepared Statements instead of EXECUTE(<String>) syntax.
- In Aurora MySQL, IF blocks must be terminated with END IF. WHILE..LOOP loops must be terminated with END LOOP.
- Unlike Oracle, Aurora MySQL auto-commit defaults to ON, be sure to set it to OFF if you want database behavior similar to Oracle.
- Similar to Oracle, collations can be defined at the server, database, and column level. They cannot be defined at the table level.
- Oracle's DELETE <Table Name> syntax, which allows omitting the FROM keyword, is invalid in Aurora MySQL. Add the FROM keyword to all DELETE statements.
- Aurora MySQL's AUTO_INCREMENT column property is similar to IDENTITY in Oracle.
- Error handling in Aurora MySQL has less features than Oracle. For special requirements, you can log or send alerts by inserting into tables or catching errors.
- Aurora MySQL does not support the MERGE statement. Use the REPLACE statement and the INSERT... ON DUPLICATE KEY UPDATE statement as alternatives.
- Unlike Oracle, You cannot concatenate strings in Aurora MySQL using the "||" operator.

- Aurora MySQL is much stricter than Oracle for statement terminators. Be sure to always use a semicolons at the end of statements.
- Aurora MySQL does not support BFILE, ROWID, UROWID data types.
- In MySQL, temporary tables are retained only for the session and only the session that created a temporary table can query it.
- MySQL does not support unused or virtual columns and there is no workaround for replacing unused columns to achieve functionality similar to virtual columns. Views and functions can be combined.
- MySQL does not support Materialized Views, use Views or summary tables instead.
- Explore AWS to locate features that can be replaced with Amazon services. They can help you maintain your database and decrease costs.
- MySQL architecture allows multiple databases in a single instance and can be useful for consolidation projects.
- Beware of control characters when copying and pasting a script to Aurora MySQL clients. Aurora MySQL is much more sensitive to control characters than Oracle and can result in frustrating syntax errors that are hard to find.

SQL & PL/SQL

Oracle Single-Row and Aggregate Functions vs. MySQL Single-Row and Aggregate Functions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Not all functions are supported by MySQL and may require manual creation

Oracle Usage

Oracle provides two main categories of built-in SQL functions based on the amount of rows used as input and generated as output.

- **Single-Row Functions** (also known as Scalar Functions) return a single result for each row of the queried table or view. They can be used with a SELECT statement in the WHERE clause, the START WITH clause, the CONNECT BY clause, and the HAVING clause. The single-row functions are divided into groups according to data type such as NUMERIC functions, CHAR functions, and DATETIME functions
- **Aggregate Functions** (also known as Group functions) are used to summarize a group of values into a single result. Examples include AVG, MIN, MAX, SUM, COUNT, LISTAGG, FIRST, and LAST.

See [MySQL Usage](#) for a comparison of Oracle and MySQL single-row functions.

Oracle 19 adds ability to eliminate duplicate items in LISTAGG function results with new DISTINCT keyword.

Oracle 19 introduces several new bitmap SQL aggregate functions (BITMAP_BUCKET_NUMBER, BITMAP_BIT_POSITION and BITMAP_CONSTRUCT_AGG) that help to speed up COUNT DISTINCT operations.

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Single-Row-Functions.html#GUID-B93F789D-B486-49FF-B0CD-0C6181C5D85C>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Aggregate-Functions.html#GUID-62BE676B-AF18-4E63-BD14-25206FEA0848>

MySQL Usage

MySQL provides an extensive list of single-row and aggregate functions. Some are similar to their Oracle counterparts (by name and functionality, or under a different name but with similar functionality). Other functions can have identical names to their Oracle counterparts, but exhibit different functionality. The Equivalent column in the table below indicates functional equivalency.

Oracle Function and definition	MySQL Function and definition	Equivalent
NUMERIC FUNCTIONS		
ABS - Absolute value of n: abs (-11.3) --> 11.3	ABS - Absolute value of n: abs (-11.3) --> 11.3	Yes
CEIL - Returns the smallest integer that is	CEIL - Returns the smallest integer that is	Yes

Oracle Function and definition	MySQL Function and definition	Equivalent
greater than or equal to n: ceil (-24.9) --> -24	greater than or equal to n: ceil (-24.9) --> -24	
FLOOR - Returns the largest integer equal to or less than n: floor (-43.7) --> -44	FLOOR - Returns the largest integer equal to or less than n: floor (-43.7) --> -44	Yes
MOD - Remainder of n2 divided by n1: mod(10,3) --> 1	MOD - Remainder of n2 divided by n1: mod(10,3) --> 1	Yes
ROUND - Returns n rounded to integer places to the right of the decimal point: round (3.49, 1) --> 3.5	ROUND - Returns n rounded to integer places to the right of the decimal point: round (3.49, 1) --> 3.5	Yes
TRUNC (Number) - Returns n1 truncated to n2 decimal places: trunc(13.5) --> 13	TRUNCATE (Number) - Returns n1 truncated to n2 decimal places: trunc(13.5) --> 13	Yes
CHARACTER FUNCTIONS		
CONCAT - Returns char1 concatenated with char2: concat('a', 1) --> a1	CONCAT - Returns char1 concatenated with char2: concat('a', 1) --> a1	Yes
LOWER / UPPER - Returns char, with all letters lowercase or uppercase: lower ('MR. Smith') --> mr. smith	LOWER / UPPER - Returns char, with all letters lowercase or uppercase: lower ('MR. Smith') --> mr. smith	Yes
LPAD / RPAD - Returns expr1, left or right padded to length n characters with the sequence of characters in expr2: LPAD('Log-1',10,'*') --> *****Log-1	LPAD / RPAD - Returns expr1, left or right padded to length n characters with the sequence of characters in expr2: LPAD('Log-1',10,'*') --> *****Log-1	Yes
REGEXP_REPLACE - Search a string for a regular expression pattern: regexp_replace ('John', '[hn].', '1') --> Jo1	Oracle REGEXP_REPLACE function can be simulated using MySQL built-in function.	No
REGEXP_SUBSTR - Extends the functionality of the SUBSTR function by searching a string for a regular expression pattern: REGEXP_SUBSTR('http://www.aws.-com/products','http://([[:alnum:]]+\.?){3,4}/?') --> http://www.aws.com/	Oracle REGEXP_REPLACE function can be simulated using MySQL built-in function.	No
REPLACE - Returns char with every occurrence of search string replaced with a replacement string: replace ('abcdef', 'abc', '123') --> 123def	REPLACE - Returns char with every occurrence of search string replaced with a replacement string: replace ('abcdef', 'abc', '123') --> 123def	Yes
LTRIM / RTRIM - Removes from the left or right end of char all of the characters that appear in set: ltrim('zzyaws', 'xyz') --> aws	LTRIM / RTRIM - Removes from the left or right end of all the space chars : ltrim(' aws') --> aws For Oracle like results you may need to combine with REPLACE proc.	Partly
SUBSTR - Return a portion of char, beginning at character position, substring length characters long: substr('John Smith', 6 ,1) --> S	SUBSTR - Return a portion of char, beginning at character position, substring length characters long: substr('John Smith', 6 ,1) --> S	Yes

Oracle Function and definition	MySQL Function and definition	Equivalent
TRIM - Trim leading or trailing characters (or both) from a character string: trim (both 'x' FROM 'xJohnxx') --> John	TRIM - Trim leading or trailing characters (or both) from a character string: trim (both 'x' FROM 'xJohnxx') --> John	Yes
ASCII - Returns the decimal representation in the database character set of the first character of char: ascii('a') --> 97	ASCII - Returns the decimal representation in the database character set of the first character of char: ascii('a') --> 97	Yes
INSTR - Search string for substring	INSTR - Search string for substring	Yes
LENGTH - Return the length of char: length ('John S.') --> 7	LENGTH - Return the length of char: length ('John S.') --> 7	Yes
REGEXP_COUNT - Returns the number of times, a pattern occurs in a source string.	Oracle REGEXP_COUNT function can be simulated using MySQL builtin function	No
REGEXP_INSTR - Search a string position for a regular expression pattern.	Oracle REGEXP_INSTR function can be simulated using MySQL built-in function.	No
DATETIME FUNCTIONS		
ADD_MONTHS - Returns the date plus integer months: add_months(sysdate, 1)	ADDDATE - MySQL can implement the same functionality using the ADDDATE function	No
CURRENT_DATE - Returns the current date in the session time zone: select current_date from dual --> 2017-01-01 13:01:01	CURRENT_DATE - MySQL CURRENT_DATE returns date with no time, use the now() or the current_timestamp function to achieve the same results: select now() --> 2017-01-01 13:01:01	Partly
CURRENT_TIMESTAMP - Returns the current date and time in the session time zone: select current timestamp from dual; --> 2017-01-01 13:01:01	CURRENT_TIMESTAMP - Returns the current date and time in the session time zone: select current timestamp from dual; --> 2017-01-01 13:01:01	Yes
EXTRACT (date part) - Returns the value of a specified datetime field from a datetime or interval expression: EXTRACT (YEAR FROM DATE '2017-03-07') --> 2017	EXTRACT (date part) - Returns the value of a specified datetime field from a datetime or interval expression: EXTRACT (YEAR FROM DATE '2017-03-07') --> 2017	Yes
LAST_DAY_MONTHS - Returns the date of the last day of the month that contains date: LAST_DAY('05-07-2018') -->05-31-2018	LAST_DAY_MONTHS - Returns the date of the last day of the month that contains date: LAST_DAY('05-07-2018') -->05-31-2018	Yes
BETWEEN - Returns the number of months between dates date1 and date2: MONTHS_BETWEEN (sysdate, sysdate-100) --> 3.25	PERIOD_DIFF - Returns the number of months between periods P1 and P2. P1 and P2 should be in the format YYYYMM. SELECT PERIOD_DIFF(201801,201703) --> 10	Partly
SYSDATE - Returns the current date and time set for the operating system on which the database server resides: select sysdate from dual --> 2017-01-01 13:01:01	SYSDATE - Returns the current date and time set for the operating system on which the database server resides: select sysdate() --> 2017-01-01 13:01:01	Yes
SYSTIMESTAMP - Returns the system date, including fractional seconds and time zone: Select systimestamp from dual --> 2017-01-	CURRENT_TIMESTAMP - Returns the current date and time in the session time zone: select current timestamp from dual; --> 2017-	Yes

Oracle Function and definition	MySQL Function and definition	Equivalent
01 13:01:01.123456 PM +00:00	01-01 13:01:01	
LOCALTIMESTAMP - Returns the current date and time in the session time zone in a value of data type TIMESTAMP: select localtimestamp from dual --> 01-JAN-17 10.01.10.123456 PM	LOCALTIMESTAMP - Returns the current date and time in the session time zone in a value of data type TIMESTAMP: select localtimestamp from dual --> 01-JAN-17 10.01.10.123456 PM	Yes
TO_CHAR (datetime) - Converts a datetime or timestamp to data type to a value of VARCHAR2 data type in the format specified by the date format: to_char(sysdate, 'DD-MON-YYYY HH24:MI:SS') --> 01-JAN-2017 01:01:01	DATE_FORMAT - DATE_FORMAT (SYSDATE(), '%Y-%m-%d %H:%i:%s')	Yes
TRUNC (date) - Returns a date with the time portion of the day truncated to the unit specified by the format model: trunc(sys-timestamp) --> 2017-01-01 00:00:00	Oracle TRUNC function can be simulated using MySQL built-in function	No
ENCODING AND DECODING FUNCTIONS		
DECODE - Compares expr to each search value one by one using the functionality of an IF-THEN-ELSE statement	CASE - Compares expr to each search value one by one	No
DUMP - Returns a VARCHAR2 value containing the data type code, length in bytes, and internal representation of expr.	N/A	No
ORA_HASH - Computes a hash value for a given expression.	SHA - Calculates an SHA-1 160-bit checksum for the string	No
NULL FUNCTIONS		
CASE - The CASE statement chooses from a sequence of conditions and runs a corresponding statement: CASE WHEN condition THEN result [WHEN ...] [ELSE result] END	CASE - The CASE statement chooses from a sequence of conditions and runs a corresponding statement: CASE WHEN condition THEN result [WHEN ...] [ELSE result] END	Yes
COALESCE - Returns the first non-null expr in the expression list: coalesce (null, 'a', 'b') --> a	COALESCE - Returns the first non-null expr in the expression list: coalesce (null, 'a', 'b') --> a	Yes
NULLIF - Compares expr1 and expr2. If they are equal, the function returns null. If they are not equal, the function returns expr1: NULLIF ('a', 'b') --> a	NULLIF - Compares expr1 and expr2. If they are equal, then the function returns null. If they are not equal, then the function returns expr1: NULLIF ('a', 'b') --> a	Yes
NVL - Replace null (returned as a blank) with a string in the results of a query: NVL (null, 'a') --> a	IFNULL - Replace null (returned as a blank) with a string in the results of a query: IFNULL (null, 'a') --> a	No
NVL2 - Determine the value returned by a query based on whether a specified expression is null or not null.	CASE - The CASE statement chooses from a sequence of conditions and runs a corresponding statement: CASE WHEN condition THEN result [WHEN ...] [ELSE result] END	No



Oracle Function and definition	MySQL Function and definition	Equivalent
ENVIRONMENT AND IDENTIFIER FUNCTIONS		
SYS_GUID - Generates and returns a globally unique identifier (RAW value) made up of 16 bytes: select sys_guid() from dual --> 5A280ABA8C76201EE0530100007FF691	UUID + REPLACE - REPLACE(UUID(), '-', '')	No
UID - Returns an integer that uniquely identifies the session user (the user who logged on): select uid from dual --> 84	N/A	No
USER - Returns the name of the session user: select user from dual	USER - Returns the name of the session user and source machine: select USER()	No
USERENV - Returns information about the current session using parameters: SELECT USERENV('LANGUAGE') "Language" FROM DUAL	SHOW SESSION VARIABLES - The statement displays the system variable values that are in effect for the current connection: myshow SESSION VARIABLES LIKE 'collation_connection';	No
CONVERSION FUNCTIONS		
CAST - Converts one built-in data type or collection-typed value into another built-in data type or collection-typed value: cast ('10' as int) + 1 --> 11	CAST - Converts one built-in data type or collection-typed value into another built-in data type or collection-typed value: CAST('10' AS UNSIGNED)+1	Yes
CONVERT - Converts a character string from a one-character set to another: select convert ('Ä Ê Í Õ Ø A B C D E ', 'US7ASCII', 'WE8ISO8859P1') from dual	CONVERT - Converts a character string from a one-character set to another: select convert ('Ä Ê Í Õ Ø A B C D E ' USING utf8)	Yes
TO_CHAR (string / numeric) - Converts NCHAR, NVARCHAR2, CLOB, or NCLOB data to the database character set: select to_char('01234') from dual --> 01234	FORMAT - Converts string data to the database character set: FORMAT('01234', 0) --> 01234	No
TO_DATE - Converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type: to_date('01Jan2017', 'DDMonYYYY') --> 01-JAN-17	STR_TO_DATE - Converts string data type to a value of DATE data type: SELECT STR_TO_DATE('01Jan2017', '%d%M%Y')	No
TO_NUMBER - Converts expr to a value of NUMBER data type: to_number('01234') --> 1234 or to_number('01234', '99999') --> 1234	N/A	No
AGGREGATE FUNCTIONS		
AVG - Returns average value of expr: select avg(salary) from employees	AVG - Returns average value of expr: select avg(salary) from employees	Yes
COUNT - Returns the number of rows returned by the query: select count(*) from employees	COUNT - Returns the number of rows returned by the query: select count(*) from employees	Yes

Oracle Function and definition	MySQL Function and definition	Equivalent
LISTAGG - Orders data within each group specified in the ORDER BY clause and then concatenates the values of the measure column: select listagg(first name, ' ,') within group (order by customerid) from customer	GROUP_CONCAT - Orders data within each group specified in the ORDER BY clause and then concatenates the values of the measure column: select GROUP_CONCAT(firstname order by customerid) from customer	No
MAX - Returns the maximum value of expr: select max(salary) from employees	MAX - Returns maximum value of expr: select max(salary) from employees	Yes
MIN - Returns the minimum value of expr: select min(salary) from employees	MIN - Returns minimum value of expr: select min(salary) from employees	Yes
SUM - Returns the sum of values of expr: select sum(salary) from employees	SUM - Returns the sum of values of expr: select sum(salary) from employees	Yes
Top-N Query Oracle 12c		
FETCH - Retrieves rows of data from the result set of a multi-row query: select * from customer fetch first 10 rows only	LIMIT - Retrieve just a portion of the rows that are generated by the rest of the query: select * from customer LIMIT 10	Yes

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/string-functions.html>
- <https://dev.mysql.com/doc/refman/5.7/en/group-by-functions-and-modifiers.html>
- <https://dev.mysql.com/doc/refman/5.7/en/numeric-functions.html>

Oracle Create Table as Select (CTAS) vs. MySQL Create Table as Select (CTAS)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	

Oracle Usage

The Create Table As Select (CTAS) statement creates a new table based on an existing table. It copies the table DDL definitions (column names and column datatypes) and data to a new table. The new table is populated from the columns specified in the SELECT statement, or all columns if you use SELECT * FROM. You can filter specific data using the WHERE and AND statements. Additionally, you can create a new table having a different structure using joins, GROUP BY, and ORDER BY.

Examples

Create a table based on an existing table and include data from all columns.

```
CREATE TABLE EMPS
AS
SELECT * FROM EMPLOYEES;
```

Create a table based on an existing table with select columns.

```
CREATE TABLE EMPS
AS
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY FROM EMPLOYEES
ORDER BY 3 DESC;
```

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.htm-#GUID-F9CE0CC3-13AE-4744-A43C-EAC7A71AAAB6>

MySQL Usage

MySQL conforms to the ANSI/SQL standard for CTAS functionality and is compatible with an Oracle CTAS statement. For MySQL, the following CTAS standard elements are optional:

- The standard requires parentheses around the SELECT statement; MySQL does not.
- The standard requires the WITH [NO] DATA clause; MySQL does not.

Examples

Create a table based on an existing table and include data from all columns.



```
CREATE TABLE EMPS AS SELECT * FROM EMPLOYEES;
```

Create a table based on an existing table with select columns.

```
CREATE TABLE EMPS AS SELECT EMPLOYEE_ID, FIRST_NAME, SALARY FROM EMPLOYEES ORDER BY 3  
DESC;
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.html#GUID-F9CE0C4744-A43C-EAC7A71AAAB6>

Oracle Common Table Expression (CTE) vs. MySQL Common Table Expressions (CTE)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Common Table Expressions	CTE is not supported in MySQL, work-around available

Oracle Usage

Common Table Expressions (CTE) provide a way to implement the logic of sequential code or to reuse code. You can define a named subquery and then use it multiple times in different parts of a query statement.

A CTE is implemented using a WITH clause, which is part of the ANSI SQL-99 standard and has existed in Oracle since version 9.2. CTE usage is similar to an inline view or a temporary table. Its main purpose is to reduce query statement repetition and make complex queries simpler to read and understand.

Syntax

```
WITH <subquery name> AS (<subquery code>)[...]
SELECT <Select list> FROM <subquery name>;
```

Examples

Create a subquery of the employee count for each department and then use the result set of the CTE in a query.

```
WITH DEPT_COUNT
(DEPARTMENT_ID, DEPT_COUNT) AS
(SELECT DEPARTMENT_ID, COUNT(*)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID)

SELECT E.FIRST_NAME || ' ' || E.LAST_NAME AS EMP_NAME,
D.DEPT_COUNT AS EMP_DEPT_COUNT
FROM EMPLOYEES E JOIN DEPT_COUNT D
USING (DEPARTMENT_ID)
ORDER BY 2;
```

MySQL Usage

Aurora MySQL 5.7 does not support Common Table Expressions (CTE).

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports common table expressions both non-recursive and recursive. Common table expressions enable use of named temporary result sets implemented by permitting a WITH clause preceding SELECT statements and certain other statements. For more information see Section 13.2.15 WITH (Common Table Expressions). As of

MySQL 8.0.19, the recursive SELECT part of a recursive common table expression (CTE) supports a LIMIT clause. LIMIT with OFFSET is also supported. See [Recursive Common Table Expressions for more information](#).

Migration Considerations

As a workaround, use VIEWS or derived tables in place of non-recursive CTEs.

Since non-recursive CTEs are more convenient for readability and code simplification, you can convert the code to use derived tables, which are a subquery in the parent query's FROM clause. For example, replace the following CTE:

```
WITH TopCustomerOrders
( SELECT Customer, COUNT(*) AS NumOrders
  FROM Orders
  GROUP BY Customer
)
SELECT TOP 10 * FROM TopCustomerOrders ORDER BY NumOrders DESC;
```

With the following subquery:

```
SELECT *
FROM ( SELECT Customer, COUNT(*) AS NumOrders
      FROM Orders
      GROUP BY Customer ) AS TopCustomerOrders
ORDER BY NumOrders DESC
LIMIT 10 OFFSET 0;
```

When using derived tables, the derived table definition must be repeated if multiple instances are required for the query.

Converting the code for recursive CTEs is not straight forward, but you can achieve similar functionality using loops.

Examples

Replacing Non-Recursive CTEs

Use a derived table to replace non-recursive CTE functionality as follows:

Create and populate an OrderItems table.

```
CREATE TABLE OrderItems (
OrderID INT NOT NULL, Item VARCHAR(20) NOT NULL, Quantity SMALLINT NOT NULL, PRIMARY
KEY(OrderID, Item));

INSERT INTO OrderItems (OrderID, Item, Quantity) VALUES (1, 'M8 Bolt', 100), (2, 'M8
Nut', 100), (3, 'M8 Washer', 200), (3, 'M6 Washer', 100);
```

Define a derived table for TotalQty of every order and then join to the OrderItems to obtain the relative quantity for each item.

```

SELECT  O.OrderID, O.Item, O.Quantity, (O.Quantity / AO.TotalQty) * 100 AS Per-
centOfOrder
FROM    OrderItems AS O
      INNER JOIN
      ( SELECT OrderID, SUM(Quantity) AS TotalQty
        FROM OrderItems
        GROUP BY OrderID
      ) AS AO
      ON O.OrderID = AO.OrderID;

```

The example code above displays the following results:

OrderID	Item	Quantity	PercentOfOrder
1	M8 Bolt	100	100.0000000000
2	M8 Nut	100	100.0000000000
3	M8 Washer	100	33.3333333300
3	M6 Washer	200	66.6666666600

Replacing Recursive CTEs

Use recursive SQL code in stored procedures and SQL loops to replace a recursive CTEs.

Note: Stored procedure and function recursion in Aurora MySQL is disabled by default. You can set the server system variable `max_sp_recursion_depth` to a value of 1 or higher to enable recursion. However, this approach is not recommended because it may increase contention for the thread stack space.

Create and populate an Employees table.

```

CREATE TABLE Employees
( Employee VARCHAR(5) NOT NULL PRIMARY KEY, DirectManager VARCHAR(5) NULL);

INSERT INTO Employees (Employee, DirectManager)
VALUES ('John', 'Dave'), ('Jose', 'Dave'), ('Fred', 'John'), ('Dave', NULL);

```

Create an EmpHierarchy table.

```

CREATE TABLE EmpHierarchy (LVL INT, Employee VARCHAR(5), Manager VARCHAR(5));

```

Create a procedure that uses a loop to traverse the employee hierarchy.

```

CREATE PROCEDURE P()
BEGIN
DECLARE var_lvl INT;
DECLARE var_Employee VARCHAR(5);
SET var_lvl = 0;
SET var_Employee = (SELECT Employee FROM Employees WHERE DirectManager IS NULL);
INSERT INTO EmpHierarchy VALUES (var_lvl, var_Employee, NULL);
WHILE var_lvl <> -1
DO
INSERT INTO EmpHierarchy (LVL, Employee, Manager)
SELECT var_lvl + 1, Employee, DirectManager
FROM Employees
WHERE DirectManager IN (SELECT Employee FROM EmpHierarchy WHERE LVL = var_lvl);

```

```
IF NOT EXISTS (SELECT * FROM EmpHierarchy WHERE LVL = var_lvl + 1)
THEN SET var_lvl = -1;
ELSE SET var_lvl = var_lvl + 1;
END IF;
END WHILE;
END;
```

Execute the procedure.

```
CALL P ();
```

Select all records from the EmpHierarchy table.

```
SELECT * FROM EmpHierarchy;
```



Level	Employee	Manager
-----	-----	-----
0	Dave	
1	John	Dave
1	Jose	Dave
2	Fred	John

Summary

Oracle	Aurora MySQL	Comments
Non recursive CTE	Derived table	For multiple instances of the same table, the derived table definition subquery must be repeated.
Recursive CTE	Loop inside a stored procedure or stored function.	

For more information, [see https://dev.mysql.com/doc/refman/5.7/en/with.html](https://dev.mysql.com/doc/refman/5.7/en/with.html)

Oracle Sequences and Identity Columns vs. MySQL Sequences and AUTO INCREMENT Columns

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Sequences and Identity	MySQL does not support sequences, Identity column has different syntax and options

Oracle Usage

Sequences are database objects that serve as unique identity value generators that can be used, for example, to automatically generating primary key values. Oracle treats sequences as independent objects. The same sequence can generate values for multiple tables.

Sequences can be configured with multiple parameters to control their value-generating behavior. For example, the INCREMENT BY sequence parameter defines the interval between each generated sequence value. If more than one database user is generating incremented values from the same sequence, each user may encounter gaps in the generated values that are visible to them.

Oracle 18c introduced scalable sequences: a special class of sequences that are optimized for multiple concurrent session usage.

This introduce 3 new options when creating a new sequence:

- SCALE - enable the sequence scalability feature
 - EXTEND - this option will extend in additional 6 digits offset (as default) and the maximum number of digits in the sequence (maxvalue/minvalue).
 - NOEXTEND (default when using the SCALE option) - sequence value will be padded to the max value.
- NOSCALE - non-scalable sequence usage

Oracle Sequence Options

By default, the initial and increment values for a sequence are both 1, with no upper limit.

- **INCREMENT BY:** Controls the sequence interval value of the increment or decrement (if a negative value is specified). If the INCREMENT BY parameter is not specified during sequence creation, the value is set to 1. The increment cannot be assigned a value of 0.
- **START WITH:** Defines the initial value of a sequence. The default value is 1.
- **MAXVALUE | NOMAXVALUE:** Specifies the maximum limit for values generated by a sequence. It must be equal or greater than the START WITH parameter and must be greater in value than the MINVALUE parameter. The default for NOMAXVALUE is 1027 for an ascending sequence.
- **MINVALUE | NOMINVALUE:** Specifies the minimum limit for values generated by a sequence. Must be less than or equal to the START WITH parameter and must be less than the MAXVALUE parameter. The default for NOMINVALUE is -1026 for a descending sequence.

- **CYCLE | NOCYCLE:** Instructs a sequence to continue generating values despite reaching the maximum value or the minimum value. If the sequence reaches one of the defined ascending limits, it generates a new value according to the minimum value. If it reaches a descending limit, it generates a new value according to the maximum value. NOCYCLE is the default.
- **CACHE | NOCACHE:** Specifies the number of sequence values to keep cached in memory for improved performance. CACHE has a minimum value of 2. The NOCACHE parameter causes a sequence to not cache values in memory. Specifying neither CACHE nor NOCACHE will cache 20 values to memory. In the event of a database failure, all unused cached sequence values are lost and gaps in sequence values may occur.
- **SCALE | NOSCALE:** Enable the scalable sequences feature (described above)

Examples

Create a sequence.

```
CREATE SEQUENCE SEQ_EMP
START WITH 100
INCREMENT BY 1
MAXVALUE 999999999999
CACHE 20
NOCYCLE;
```

Drop a sequence.

```
DROP SEQUENCE SEQ_EMP;
```

View sequences created for the current schema/user.

```
SELECT * FROM USER_SEQUENCES;
```

Use a sequence as part of an INSERT INTO statement.

```
CREATE TABLE EMP_SEQ_TST (COL1 NUMBER PRIMARY KEY, COL2 VARCHAR2(30));
INSERT INTO EMP_SEQ_TST VALUES (SEQ_EMP.NEXTVAL, 'A');
```

COL1	COL2
-----	-----
100	A

Query the current value of a sequence.

```
SELECT SEQ_EMP.CURRVAL FROM DUAL;
```

Manually increment the value of a sequence according to the INCREMENT BY specification.

```
SELECT SEQ_EMP.NEXTVAL FROM DUAL;
```

Alter an existing sequence.

```
ALTER SEQUENCE SEQ_EMP MAXVALUE 1000000;
```

Create a scalable sequence.

```
CREATE SEQUENCE scale_seq
MINVALUE 1
MAXVALUE 9999999999
SCALE;

select scale_seq.nextval as scale_seq from dual;

NEXTVAL |
-----|
1010320001|
```

Oracle 12c Default Values Using Sequences

Beginning with Oracle 12c, you can assign a sequence to a table column with the CREATE TABLE statement and specify the NEXTVAL configuration of the sequence.

Generate DEFAULT values using sequences.

```
CREATE TABLE SEQ_TST ( COL1 NUMBER DEFAULT SEQ_1.NEXTVAL PRIMARY KEY, COL2 VARCHAR
(30));
INSERT INTO SEQ_TST(COL2) VALUES('A');
SELECT * FROM SEQ_TST;
COL1 COL2
-----|
100 A
```

Oracle 12c Session Sequences (Session/Global)

Beginning with Oracle 12c, sequences can be created as session-level or global-level. By adding the SESSION parameter to a CREATE SEQUENCE statement, the sequence is created as a session-level sequence. Optionally, the GLOBAL keyword can be used to create a global sequence to provide consistent results across sessions in the database. Global sequences are the default. Session sequences return a unique range of sequence numbers only within a session.

Create Oracle 12c SESSION and GLOBAL sequences.

```
CREATE SEQUENCE SESSION_SEQ SESSION;
CREATE SEQUENCE SESSION_SEQ GLOBAL;
```

Oracle 12c Identity Columns

Oracle 12c introduced support for automatic generation of values to populate columns in database tables. The IDENTITY type generates a sequence and associates it with a table column without the need to manually create a separate Sequence object. It relies (internally) on Sequences and can be manually configured.

Sequences can be used as an IDENTITY type, which automatically creates a sequence and associates it with the table column. The main difference is that there is no need to create a sequence manually; the IDENTITY type does that for you. An IDENTITY type is a sequence that can be configured.

Create a table with an Oracle 12c Identity Column.

```
CREATE TABLE IDENTITY_TST (
  COL1 NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 100
                                                INCREMENT BY 10),
  COL2 VARCHAR2(30));
```

Insert records using an Oracle 12c IDENTITY column (explicitly/implicitly).

```
INSERT INTO IDENTITY_TST(COL2) VALUES ('A');
INSERT INTO IDENTITY_TST(COL1, COL2) VALUES (DEFAULT, 'B');
INSERT INTO IDENTITY_TST(c0l1, c0l2) VALUES (NULL, 'C');
SELECT * FROM IDENTITY_TST;
```

COL1	COL2
-----	-----
120	A
130	B

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-SEQUENCE.htm-#GUID-E9C78A8C-615A-4757-B2A8-5E6EFB130571>

MySQL Usage

Aurora MySQL supports automatic sequence generation using the AUTO_INCREMENT column property, similar to the Oracle IDENTITY column property. It does not support table-independent sequence objects.

Any numeric column may be assigned the AUTO_INCREMENT property. To make the system generate the next sequence value, the application must not mention the relevant column's name in the insert command, in case the column was created with the NOT NULL definition then also inserting a NULL value into an AUTO_INCREMENT column will increment it. In most cases, the seed value is 1 and the increment is 1.

Client applications use the LAST_INSERT_ID function to obtain the last generated value.

Each table can have only one AUTO_INCREMENT column. The column must be explicitly indexed or be a primary key (which is indexed by default).

The AUTO_INCREMENT mechanism is designed to be used with positive numbers only. Do not use negative values because they are misinterpreted as a complementary positive value. This limitation is due to precision issues with sequences crossing a zero boundary.

There are two server parameters used to alter the default values for new AUTO_INCREMENT columns:

- **auto_increment_increment:** Controls the sequence interval.
- **auto_increment_offset:** Determines the starting point for the sequence.

To reseed the AUTO_INCREMENT value, use ALTER TABLE <Table Name> AUTO_INCREMENT = <New Seed Value>.

Migration Considerations

Since Aurora MySQL does not support table-independent SEQUENCE objects, applications that rely on its properties must use custom solutions to meet their requirements.

Aurora MySQL `AUTO_INCREMENT` can be used instead of Oracle's `IDENTITY` for most cases. For `AUTO_INCREMENT` columns, the application must explicitly `INSERT` a `NULL` or a `0`.

Note: Omitting the `AUTO_INCREMENT` column from the `INSERT` column list has the same effect as inserting a `NULL` value.

`AUTO_INCREMENT` columns must be indexed (the following section explains why) and cannot have default constraints assigned to the same column. There is a critical difference between `IDENTITY` and `AUTO_INCREMENT` in the way the sequence values are maintained upon service restart. Application developers must be aware of this difference.

Sequence Value Initialization

Oracle stores the `IDENTITY` metadata in system tables on disk. Although some values may be cached and are lost when the service is restarted, the next time the server restarts, the sequence value continues after the last block of values that was assigned to cache. If you run out of values, you can explicitly set the sequence value to start the cycle over. As long as there are no key conflicts, it can be reused after the range has been exhausted.

In Aurora MySQL, an `AUTO_INCREMENT` column for a table uses a special auto-increment counter to assign new values for the column. This counter is stored in cache memory only and is not persisted to disk. After a service restart, and when Aurora MySQL encounters an `INSERT` to a table that contains an `AUTO_INCREMENT` column, it issues an equivalent to the following statement:

```
SELECT MAX(<Auto Increment Column>) FROM <Table Name> FOR UPDATE;
```

Note: The `FOR UPDATE` `CLAUSE` is required to maintain locks on the column until the read completes.

Aurora MySQL then increments the value retrieved by the statement above and assigns it to the in-memory auto-increment counter for the table.

By default, the value is incremented by one. You can change this default using the `auto_increment_increment` configuration setting. If the table has no values, Aurora MySQL uses the value 1. You can change the default using the `auto_increment_offset` configuration setting.

Every server restart effectively cancels any `AUTO_INCREMENT = <Value>` table option in `CREATE TABLE` and `ALTER TABLE` statements.

Unlike Oracle `IDENTITY` columns, which by default do not allow inserting explicit values, Aurora MySQL allows explicit values to be set. If a row has an explicitly specified `AUTO_INCREMENT` column value and the value is greater than the current counter value, the counter is set to the specified column value.

Examples

Create a table with an `AUTO_INCREMENT` column.

```
CREATE TABLE MyTable (Col1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY, Col2 VARCHAR(20) NOT NULL);
```

Insert `AUTO_INCREMENT` Values.

```
INSERT INTO MyTable (Col2) VALUES ('AI column omitted');
```

```
INSERT INTO MyTable (Col1, Col2) VALUES (NULL, 'Explicit NULL');
```

```
INSERT INTO MyTable (Col1, Col2) VALUES (10, 'Explicit value');
```

```
INSERT INTO MyTable (Col2) VALUES ('Post explicit value');
```

```
SELECT * FROM MyTable;
```

```
Col1    Col2
----    -
1       AI column omitted
2       Explicit NULL
10      Explicit value
11      Post explicit value
```

Reseed AUTO_INCREMENT.

```
ALTER TABLE MyTable AUTO_INCREMENT = 30;
```

```
INSERT INTO MyTable (Col2) VALUES ('Post ALTER TABLE');
```

```
SELECT * FROM MyTable;
```

```
Col1    Col2
----    -
1       AI column omitted
2       Explicit NULL
10      Explicit value
11      Post explicit value
30      Post ALTER TABLE
```

Summary

The following table identifies similarities, differences, and key migration considerations.



Feature	Oracle	Aurora MySQL	Comments
Create a table	CREATE TABLE IDENTITY_TST (COL1 NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 10), COL2 VARCHAR2 (30));	CREATE TABLE AUTO_TST (COL1 INT AUTO_INCREMENT PRIMARY KEY, COL2 VARCHAR(30));	
set the starting number	see table create example	ALTER TABLE AUTO_TST AUTO_INCREMENT = 100; OR setting the auto_increment_offset parameter	
set the interval	see table create example	Setting the auto_increment_increment parameter	
additional permitted values	DEFAULT, NULL	none	

Feature	Oracle	Aurora MySQL	Comments
Independent SEQUENCE object	CREATE SEQUENCE	Not supported	
Automatic enumerator column property	IDENTITY	AUTO_INCREMENT	
Reseed sequence value	Recreate the sequence	ALTER TABLE	
Column restrictions	Numeric	Numeric, indexed, and no DEFAULT	
Controlling seed and interval values	CREATE/ALTER TABLE	auto_increment_increment auto_increment_offset	Aurora MySQL settings are global and cannot be customized for each column as with Oracle.
Sequence setting initialization	Maintained through service restarts	Re-initialized every service restart	See the Sequence Value Initialization section.
Explicit values to column	not supported.	Supported	Aurora MySQL requires explicit NULL or 0 to trigger sequence value assignment. Inserting an explicit value larger than all others reinitializes the sequence.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/example-auto-increment.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-table.html>
- <https://dev.mysql.com/doc/refman/5.7/en/innodb-auto-increment-handling.html#innodb-auto-increment-initialization>

Oracle Insert From Select vs. MySQL Insert From Select

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	ERROR LOG and subquery options are not supported

Oracle Usage

You can insert multiple records into a table from another table using the INSERT FROM SELECT statement, which is a derivative of the basic INSERT statement. The column ordering and data types must match between the target and the source tables.

Examples

Simple INSERT FROM SELECT (Explicit):

```
INSERT INTO EMPS (EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID) SELECT EMPLOYEE_ID,
FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY > 10000;
```

Simple INSERT FROM SELECT (Implicit):

```
INSERT INTO EMPS
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY > 10000;
```

This example produces the same result as the preceding example but uses a subquery in the DML_table_expression_clause.

```
INSERT INTO
(SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID FROM EMPS)
VALUES (120, 'Kenny', 10000, 90);
```

Log errors with the Oracle error_logging_clause.

```
ALTER TABLE EMPS ADD CONSTRAINT PK_EMP_ID PRIMARY KEY(employee_id);
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('EMPS', 'ERRLOG');
INSERT INTO EMPS
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY > 10000
LOG ERRORS INTO errlog ('Cannot Perform Insert') REJECT LIMIT 100;

0 rows inserted
```

When inserting an existing EMPLOYEE ID into the EMPS table, the insert does not fail because the invalid records are redirected to the ERRLOG table.

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/INSERT.html#GUID-903F8043-0254-4EE9-ACC1-CB8AC0AF3423>

MySQL Usage

MySQL is compatible with the Oracle INSERT FROM SELECT syntax except for a few features specific to Oracle. For example, the conditional_insert_clause (ALL | FIRST | ELSE). MySQL does not support the Oracle error_logging_clause feature. Generally, you can use ON DUPLICATE KEY UPDATE to handle duplicate rows.

Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name
      [PARTITION (partition_name [, partition_name] ...)]
      [(col_name [, col_name] ...)]
      SELECT ...
      [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

Examples

Simple INSERT FROM SELECT (Explicit):

```
INSERT INTO EMPS (EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID) SELECT EMPLOYEE_ID,
FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY > 10000;
```

Simple Insert from Select (Implicit):

```
INSERT INTO EMPS
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY > 10000;
```

The following example is not compatible with the MySQL.


```
INSERT INTO
(SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID FROM EMPS)
VALUES (120, 'Kenny', 10000, 90);
```

The following example demonstrates using the ON DUPLICATE KEY UPDATE clause to update specific columns when a UNIQUE violation occurs.

```
INSERT INTO EMPS
SELECT * from EMPLOYEES where EMPLOYEE_ID > 10
ON DUPLICATE KEY UPDATE
EMPS.FIRST_NAME=EMPLOYEES.FIRST_NAME, EMPS.SALARY=EMPLOYEES.SALARY;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/inserts.html>

Oracle Multi-Version Concurrency Control (MVCC) vs. MySQL Multi-Version Concurrency Control (MVCC)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	ERROR LOG and subquery options are not supported

Oracle Usage

Two primary lock types exist in Oracle: exclusive locks and shared locks, which implement the following high-level locking semantics:

- Writers never block readers.
- Readers never block writers.
- Oracle never escalates locks from row to page and table level, which reduces potential deadlocks.
- Oracle allows users to issue explicit locks on a specific tables using the LOCK TABLE statement.

Lock types can be divided into four categories: DML locks, DDL locks, Explicit (Manual) data locking, and System locks. The following sections describe each category.

DML Locks

DML locks preserve the integrity of data accessed concurrently by multiple users. DML statements acquire locks automatically both on row and table levels.

- **Row Locks (TX):** Obtained on a single row of a table by one the following statements: INSERT, UPDATE, DELETE, MERGE, and SELECT ... FOR UPDATE. If a transaction obtains a row lock, a table lock is also acquired to prevent DDL modifications to the table that might cause conflicts. The lock exists until the transaction ends with a COMMIT or ROLLBACK.
- **Table Locks (TM):** When performing one of the following DML operations: INSERT, UPDATE, DELETE, MERGE, and SELECT ... FOR UPDATE, a transaction automatically acquires a table lock to prevent DDL modifications to the table that might cause conflicts if the transaction did not issue a COMMIT or ROLLBACK.

All table lock types:

- **Row Share lock (RS):** Occurs when the transaction holding the lock on the table has locked some rows in the table before updating them.
- **Row Exclusive lock (RX):** Occurs when the transaction holding the lock has updated table rows or used the SELECT ... FOR UPDATE command.
- **Share table lock (S):** One transaction locks the table and allows other transactions to query the table (exclude SELECT ... FOR UPDATE), it also allows updates only if a single transaction holds the share table lock. Multiple transactions may hold a share table lock concurrently.
- **Share Row Exclusive table lock (SRX):** Similar to S lock but with this lock, only a single transaction at a time can acquire this lock on a given table.

- **Exclusive table lock (X):** Most restrictive lock type, it allows the transaction that holds the lock an exclusive write access to the table. Only one transaction can obtain an X lock for a table.

The following table provides additional information regarding row and table locks:

Statement	Row Locks	Table Lock Mode	RS	RX	S	SRX	X
SELECT ... FROM table...	–	none	Y	Y	Y	Y	Y
INSERT INTO table...	Yes	SX	Y	Y	N	N	N
UPDATE table ...	Yes	SX	Y	Y	N	N	N
MERGE INTO table ...	Yes	SX	Y	Y	N	N	N
DELETE FROM table...	Yes	SX	Y	Y	N	N	N
SELECT ... FROM table FOR UPDATE OF...	Yes	SX	Y	Y	N	N	N
LOCK TABLE table IN...	–						
ROW SHARE MODE		SS	Y	Y	Y	Y	N
ROW EXCLUSIVE MODE		SX	Y	Y	N	N	N
SHARE MODE		S	Y	N	Y	N	N
SHARE ROW EXCLUSIVE MODE		SSX	Y	N	N	N	N
EXCLUSIVE MODE		X	N	N	N	N	N

DDL Locks

The main purpose of a DDL lock is to protect the definition of a schema object while it is modified by an ongoing DDL operation such as ALTER TABLE EMPLOYEES ADD <COLUMN>.

Explicit (Manual) Data Locking

Users have the ability to explicitly create locks to achieve transaction-level read consistency specifically for when an application requires transactional exclusive access to a resource without waiting for other transactions to complete. Explicit data locking can be performed at the transaction level or the session level:

Transaction Level:

- SET TRANSACTION ISOLATION LEVEL
- LOCK TABLE
- SELECT ... FOR UPDATE

Session Level:

- ALTER SESSION SET ISOLATION LEVEL

System Locks

System locks include Latches, Mutexes, and internal locks.

Examples

Explicitly lock data using the LOCK TABLE command.

```
-- Session 1
LOCK TABLE EMPLOYEES IN EXCLUSIVE MODE;

-- Session 2
UPDATE EMPLOYEES
SET SALARY=SALARY+1000
WHERE EMPLOYEE_ID=114;

-- Session 2 waits for session 1 to COMMIT or ROLLBACK
```

Explicitly lock data using the SELECT... FOR UPDATE command. Oracle obtains exclusive row-level locks on all the rows identified by the SELECT FOR UPDATE statement.

```
-- Session 1
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=114 FOR UPDATE;

-- Session 2
UPDATE EMPLOYEES
SET SALARY=SALARY+1000
WHERE EMPLOYEE_ID=114;

-- Session 2 waits for session 1 to COMMIT or ROLLBACK
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Automatic-Locks-in-DDL-Operations.html#GUID-84D392A3-94EC-444D-950F-7829DBCD43EE>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Automatic-Locks-in-DML-Operations.html#GUID-3D57596F-8B73-4C80-8F4D-79A12F781EFD>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Automatic-and-Manual-Locking-Mechanisms-During-SQL-Operations.html#GUID-0304C4AA-BD28-4C2A-B7F5-267532FB9499>

MySQL Usage

When using InnoDB, MySQL provides various lock modes to control concurrent access to data in tables. Data consistency is maintained using a Multi-Version Concurrency Control (MVCC) mechanism. Most MySQL commands automatically acquire locks of appropriate modes to ensure that referenced tables are not dropped or modified in incompatible ways while the command executes.

The MVCC mechanism prevents viewing inconsistent data produced by concurrent transactions performing updates on the same rows. MVCC provides strong transaction isolation for each database session and minimizes lock-contention in multi-user environments.

- Similar to Oracle, MVCC locks acquired for querying (reading) data do not conflict with locks acquired for writing data. Reads never block writes and writes never blocks reads.
- Similar to Oracle, MySQL does not escalate locks to table-level such as when an entire table is locked for writes when a certain threshold of row locks is exceeded.

InnoDB uses three additional fields for each row:

- **DB_TRX_ID**: Indicates the transaction identifier for the last transaction that inserted or updated the row.
- **DB_ROLL_PTR**: Points to an undo log record written to the rollback segment.
- **DB_ROW_ID**: Contains a row ID that increases monotonically as new rows are inserted.

Implicit and Explicit Transactions (Auto-Commit Behavior)

Unlike Oracle, MySQL uses auto-commit for transactions by default. However, there are two options to support explicit transactions, which are similar to the default behavior in Oracle (non-auto-commit):

- Use the `START TRANSACTION` statement and then `COMMIT` or `ROLLBACK`.
- Set `AUTOCOMMIT` to `OFF` at the session level.

With explicit transactions:

- Users can explicitly issue a lock similar to the `LOCK TABLE` statement in Oracle.
- `SELECT... FOR UPDATE` is supported.

Unlike Oracle there are only two types of table-level locks when using the `LOCK TABLE` command: Read Lock and Write Lock.

READ [LOCAL] lock (shared S lock)

- The session that holds the lock can only read the table.
- Multiple sessions can acquire a READ lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a READ lock.
- For InnoDB tables, READ LOCAL is the same as READ.

[LOW_PRIORITY] WRITE lock (exclusive X lock)

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the WRITE lock is held.
- The LOW_PRIORITY modifier is deprecated and has no effect.

For row-level locking:

- **Intention shared IS lock:** Indicates that a transaction intends to set a shared lock.
- **Intention exclusive IX lock:** Indicates that a transaction intends to set a exclusive lock.

	X	IX	S	IS
X	Not permitted	Not permitted	Not permitted	Not permitted
IX	Not permitted	Permitted	Not permitted	Permitted
S	Not permitted	Not permitted	Permitted	Permitted
IS	Not permitted	Permitted	Permitted	Permitted

Records lock

A record lock is a lock on an index record. For example, the `SELECT id FROM emps WHERE id = 50 FOR UPDATE` query prevents any other transaction from inserting, updating, or deleting rows where the value of `emps.id` is 50.

Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses it for record locking.

Gaps lock

A gap lock is a lock on a gap between index records, before the first index record, or after the last index record. For example, `SELECT id FROM emps WHERE id BETWEEN 50 and 80 FOR UPDATE` prevents other transactions from inserting a value of 60 into the `emps.id` column whether or not there was already any value in the column because the gaps between all existing values in the range are locked.

Transaction-level locking

- SET TRANSACTION ISOLATION LEVEL
- LOCK TABLE
- SELECT ... FOR UPDATE

LOCK TABLE Syntax

```
LOCK TABLES
tbl_name [[AS] alias] lock_type [, tbl_name [[AS] alias] lock_type] ...

lock_type:
READ [LOCAL] | [LOW_PRIORITY] WRITE
```

MySQL Deadlocks

Deadlocks occur when two or more transactions acquired locks on each other's process resources (table or row). MySQL can detect deadlocks automatically and resolve the event by aborting one of the transactions and allowing the other transaction to complete.

Examples

Obtain an explicit lock on a table using the LOCK TABLE command.

```
-- Session 1
START TRANSACTION;
LOCK TABLE EMPLOYEES WRITE;

-- Session 2
UPDATE EMPLOYEES
SET SALARY=SALARY+1000
WHERE EMPLOYEE_ID=114;

-- Session 2 waits for session 1 to COMMIT or ROLLBACK
```

Set an explicit lock using the SELECT... FOR UPDATE command. MySQL obtains exclusive row-level locks on rows referenced by the SELECT FOR UPDATE statement. It must be executed inside a transaction.

```
-- Session 1
START TRANSACTION;
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=114 FOR UPDATE;

-- Session 2
UPDATE EMPLOYEES
SET SALARY=SALARY+1000
WHERE EMPLOYEE_ID=114;

-- Session 2 waits for session 1 to COMMIT or ROLLBACK
```



Summary

Description	Oracle	MySQL
“Dictionary” tables to obtain information about locks	v\$lock; v\$locked_object; v\$session_blockers;	Show open tables where in_use = 1;
Lock a table	BEGIN; LOCK TABLE employees IN SHARE ROW EXCLUSIVE MODE;	LOCK TABLE employees READ
Explicit Locking	SELECT * FROM employees WHERE employee_id=102 FOR UPDATE;	SELECT * FROM employees WHERE employee_id=102 FOR UPDATE;
Explicit Locking , options	SELECT...FOR UPDATE	SELECT...FOR UPDATE

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/innodb-multi-versioning.html>
- <https://dev.mysql.com/doc/refman/5.7/en/lock-tables.html>
- <https://dev.mysql.com/doc/refman/5.7/en/set-transaction.htmlx>

Oracle Merge vs. MySQL Merge

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MERGE is not supported, workaround available

Oracle Usage

The MERGE statement provides a means to specify single SQL statements that conditionally perform INSERT, UPDATE, or DELETE operations on a target table—a task that would otherwise require multiple logical statements.

The MERGE statement selects record(s) from the source table and then, by specifying a logical structure, automatically performs multiple DML operations on the target table. Its main advantage is to help avoid the use of multiple inserts, updates or deletes. It is important to note that MERGE is a deterministic statement. That is, once a row has been processed by the MERGE statement, it cannot be processed again using the same MERGE Statement. MERGE is also sometimes known as UPSERT.

Examples

Use MERGE to insert or update employees who are entitled to a bonus (by year):

```
CREATE TABLE EMP_BONUS (EMPLOYEE_ID NUMERIC, BONUS_YEAR VARCHAR2(4),
SALARY NUMERIC, BONUS NUMERIC, PRIMARY KEY (EMPLOYEE_ID, BONUS_YEAR));
```

```
MERGE INTO EMP_BONUS E1
USING (SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID
FROM EMPLOYEES) E2 ON (E1.EMPLOYEE_ID = E2.EMPLOYEE_ID) WHEN MATCHED THEN
UPDATE SET E1.BONUS = E2.SALARY * 0.5
DELETE WHERE (E1.SALARY >= 10000)
WHEN NOT MATCHED THEN
INSERT (E1.EMPLOYEE_ID, E1.BONUS_YEAR, E1.SALARY, E1.BONUS)
VALUES (E2.EMPLOYEE_ID, EXTRACT(YEAR FROM SYSDATE), E2.SALARY,
E2.SALARY * 0.5)
WHERE (E2.SALARY < 10000);
```

```
SELECT * FROM EMP_BONUS;
```

```
EMPLOYEE_ID BONUS_YEAR SALARY BONUS
-----
103 2017 9000 4500
104 2017 6000 3000
105 2017 4800 2400
106 2017 4800 2400
107 2017 4200 2100
111 2017 7700 3850
112 2017 7800 3900
113 2017 6900 3450
115 2017 3100 1550
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/MERGE.html#GUID-5692CCB7-24D9-4C0E-81A7-A22436DC968F>

MySQL Usage

Aurora MySQL does not support the MERGE statement. However, it provides two other statements for merging data: REPLACE, and INSERT... ON DUPLICATE KEY UPDATE.

REPLACE deletes a row and inserts a new row if a duplicate key conflict occurs. INSERT... ON DUPLICATE KEY UPDATE performs an in-place update. Both REPLACE and ON DUPLICATE KEY UPDATE rely on an existing primary key and unique constraints. It is not possible to define custom MATCH conditions as with Oracle's MERGE statement.

REPLACE provides a function similar to INSERT. The difference is that REPLACE first deletes an existing row if a duplicate key violation for a PRIMARY KEY or UNIQUE constraint occurs.

REPLACE is a MySQL extension that is not ANSI compliant. It either performs only an INSERT when no duplicate key violations occur, or it performs a DELETE and then an INSERT if violations occur.

Syntax

```
REPLACE [INTO] <Table Name> (<Column List>) VALUES v(<Values List>)
```

```
REPLACE [INTO] <Table Name> SET <Assignment List: ColumnName = VALUE...>
```

```
REPLACE [INTO] <Table Name> (<Column List>) SELECT ...
```

INSERT ... ON DUPLICATE KEY UPDATE

The ON DUPLICATE KEY UPDATE clause of the INSERT statement acts as a dual DML hybrid. Similar to REPLACE, it executes the assignments in the SET clause instead of raising a duplicate key error. ON DUPLICATE KEY UPDATE is a MySQL extension that is not ANSI compliant.

```
INSERT [INTO] <Table Name> [<Column List>] VALUES (<Value List> ON DUPLICATE KEY  
<Assignment List: ColumnName = Value...>
```

```
INSERT [INTO] <Table Name> SET <Assignment List: ColumnName = Value...> ON DUPLICATE  
KEY UPDATE <Assignment List: ColumnName = Value...>
```

```
INSERT [INTO] <Table Name> [<Column List>] SELECT ... ON DUPLICATE KEY UPDATE <Assign-  
ment List: ColumnName = Value...>
```

Migration Considerations

Neither REPLACE nor INSERT ... ON DUPLICATE KEY UPDATE provide a full functional replacement for Oracle's MERGE statement. The key differences are:

- Key violation conditions are mandated by the primary key or unique constraints that exist on the target table. They can not be defined using an explicit predicate.

- There is no alternative for the WHEN NOT MATCHED BY SOURCE clause.
- There is no alternative for the OUTPUT clause.

The key difference between REPLACE and INSERT ON DUPLICATE KEY UPDATE is that with REPLACE, the violating row is deleted or attempted to be deleted. If foreign keys are in place, the DELETE operation may fail, which may fail the entire transaction.

For INSERT ... ON DUPLICATE KEY UPDATE, the update is performed on the existing row in place without attempting to delete it.

It should be straightforward to replace most MERGE statements with either REPLACE or INSERT... ON DUPLICATE KEY UPDATE. Alternatively, break down the operations into their constituent INSERT, UPDATE, and DELETE statements.

Examples

Use REPLACE to create a simple one-way, two-table sync.

```
CREATE TABLE SourceTable (Col1 INT NOT NULL PRIMARY KEY, Col2 VARCHAR(20) NOT NULL);
CREATE TABLE TargetTable (Col1 INT NOT NULL PRIMARY KEY, Col2 VARCHAR(20) NOT NULL);
```

```
INSERT INTO SourceTable (Col1, Col2) VALUES (2, 'Source2'), (3, 'Source3'), (4,
'Source4');
INSERT INTO TargetTable (Col1, Col2) VALUES (1, 'Target1'), (2, 'Target2'), (3, 'Tar-
get3');
```

```
REPLACE INTO TargetTable(Col1, Col2) SELECT Col1, Col2 FROM SourceTable;
```

```
SELECT * FROM TargetTable;
```

Col1	Col2
1	Target1
2	Source2
3	Source3
4	Source4

Create a conditional two-way sync using NULL for "no change" and DELETE from target when not found in source.

```
TRUNCATE TABLE SourceTable;
```

```
INSERT INTO SourceTable(Col1, Col2) VALUES (3, NULL), (4, 'NewSource4'), (5,
'Source5');
DELETE FROM TargetTable WHERE Col1 NOT IN (SELECT Col1 FROM SourceTable);
```

```
INSERT INTO TargetTable (Col1, Col2)
SELECT Col1, Col2
FROM SourceTable AS SRC
WHERE SRC.Col1 NOT IN (SELECT Col1 FROM TargetTable);
```

```
UPDATE TargetTable AS TGT
SET Col2 = (SELECT COALESCE(SRC.Col2, TGT.Col2)
```

```
FROM SourceTable AS SRC WHERE SRC.Col1 = TGT.Col1)
WHERE TGT.Col1 IN (SELECT Col1 FROM SourceTable);
```

```
SELECT *
FROM TargetTable;
```

```
Col1    Col2
-----  -----
3        Source3
4        NewSource4
5        Source5
```

Summary



The following table describes similarities, differences, and key migration considerations.

Oracle MERGE feature	Migrate to Aurora MySQL	Comments
Define source set in USING clause	Define source set in a SELECT query or in a table.	
Define logical duplicate key condition with an ON predicate	Duplicate key condition mandated by primary key and unique constraints on target table.	
WHEN MATCHED THEN UPDATE	REPLACE or INSERT... ON DUPLICATE KEY UPDATE	When using REPLACE, the violating row is deleted, or attempted to be deleted. If there are foreign keys in place, the DELETE operation may fail, which may fail the entire transaction. With INSERT ... ON DUPLICATE KEY UPDATE, the updated is performed on the existing row in place, without attempting to delete it.
WHEN MATCHED THEN DELETE	DELETE FROM Target WHERE Key IN (SELECT Key FROM Source)	
WHEN NOT MATCHED THEN INSERT	REPLACE or INSERT... ON DUPLICATE KEY UPDATE	See above comment.

For more information, see:

- <https://dev.mysql.com/doc/refman/5.7/en/replace.html>
- <https://dev.mysql.com/doc/refman/5.7/en/insert-on-duplicate.html>

Oracle OLAP Functions vs. MySQL Window Functions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	<ul style="list-style-type: none"> • GREATEST and LEAST functions might get different results • CONNECT BY is not supported by MySQL, workaround available

Oracle Usage

Oracle OLAP functions extend the functionality of standard SQL analytic functions by providing capabilities to compute aggregate values based on a group of rows. You can apply the OLAP functions to logically partitioned sets of results within the scope of a single query expression. OLAP functions are usually used in combination with Business Intelligence reports and analytics. They can help boost query performance as an alternative to achieving the same result using more complex non-OLAP SQL code.

Common Oracle OLAP Functions:

Function Type	Related Functions
Aggregate	average_rank, avg, count, dense_rank, max, min, rank, sum
Analytic	average_rank, avg, count, dense_rank, lag, lag_variance, lead_variance_percent, max, min, rank, row_number, sum, percent_rank, cume_dist, ntile, first_value, last_value
Hierarchical	hier_ancestor, hier_child_count, hier_depth, hier_level, hier_order, hier_parent, hier_top
Lag	lag, lag_variance, lag_variance_percent, lead, lead_variance, lead_variance_percent
OLAP DML	olap_dml_expression
Rank	average_rank, dense_rank, rank, row_number

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Functions.html>

MySQL Usage

Some Oracle OLAP functions are aggregative functions in Aurora MySQL. For a description of the MySQL equivalent functions, see [Single-Row and Aggregate Functions](#).

Other OLAP functions can be replaced with window functions, which are currently not available in Aurora MySQL.

Aurora MySQL version 5.7 does not support window functions.

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports window functions that for each row from a query perform a calculation using rows related to that row. These include functions such as RANK(), LAG() and NTILE(). In addition several existing aggregate functions now can be used as window functions (for example SUM() and AVG()). For more information see [Section 12.21 Window Functions](#).

Migration Considerations

As a temporary workaround, rewrite the code to remove the use of Window Functions, and revert to using more traditional SQL code solutions.

In most cases, you can find an equivalent SQL query, although it may be less optimal in terms of performance, simplicity, and readability. See the examples below for migrating window functions to code that uses correlated subqueries.

Note: You may want to archive the original code and then reuse it in the future when Aurora MySQL is upgraded to version 8. The documentation for version 8 indicates the window function syntax is ANSI compliant and will be compatible with Oracle's PL\SQL syntax.

For more information about the upcoming version 8 support for window functions, see <https://dev.mysql.com/doc/refman/5.7/en/window-functions.html>

Examples

The following examples demonstrate ANSI SQL compliant subquery solutions:

Create and populate an `OrderItems` table.

```
CREATE TABLE OrderItems (
  OrderID INT NOT NULL, Item VARCHAR(20) NOT NULL, Quantity SMALLINT NOT NULL, PRIMARY
  KEY(OrderID, Item));

INSERT INTO OrderItems (OrderID, Item, Quantity) VALUES (1, 'M8 Bolt', 100), (2, 'M8
  Nut', 100), (3, 'M8 Washer', 200);
```

Rank items based on ordered quantity (workaround for the window ranking function).



```
SELECT  Item, Quantity,
        (SELECT COUNT(*) FROM OrderItems AS OI2 WHERE OI.Quantity > OI2.Quantity) + 1
  AS QtyRank
FROM    OrderItems AS OI;
```

Calculate the grand total (workaround for a partitioned window aggregate function) .

```
SELECT  Item, Quantity,
        (SELECT SUM(Quantity) FROM OrderItems AS OI2 WHERE OI2.OrderID = OI.OrderID)
  AS TotalOrderQty
FROM    OrderItems AS OI;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/window-function-descriptions.html>

Oracle Transaction Model vs. MySQL Transactions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Transaction Model	MySQL default isolation level is REPEATABLE READ Nested transactions are not supported in MySQL

Oracle Usage

Database transactions are logical, atomic units of processing containing one or more SQL statements that may run concurrently alongside other transactions. The primary purpose of a transaction is to ensure the ACID model is enforced.

- **Atomicity:** All statements in a transaction are processed as one logical unit, or none are processed. If a single part of a transaction fails, the entire transaction is aborted and no changes are persisted (“all or nothing”).
- **Consistency:** All data integrity constraints are checked and all triggers are processed before a transaction is processed. If any of the constraints are violated, the entire transaction fails.
- **Isolation:** One transaction is not affected by the behavior of other concurrent transactions. The effect of a transaction is not visible to other transactions until the transaction is committed.
- **Durability:** Once a transaction commits, its results are not lost regardless of subsequent failures. After a transaction completes, changes made by committed transactions are permanent. The database ensures that committed transactions cannot be lost.

Database Transaction Isolation Levels

The ANSI/ISO SQL standard (SQL92) defines four levels of isolation. Each level provides a different approach for handling concurrent execution of database transactions. Transaction isolation levels manage the visibility of changed data as seen by other running transactions. In addition, when accessing the same data with several concurrent transactions, the selected level of transaction isolation affects the way different transactions interact. For example, if a bank account is shared by two individuals, what happens if both parties attempt to perform a transaction on the shared account at the same time? One checks the account balance while the other withdraws money. Oracle supports the following isolation levels:

- **Read Uncommitted Isolation Level:** A currently processed transaction can see uncommitted data made by the other transaction. If a rollback is performed, all data is restored to its previous state.
- **Read Committed Isolation Level:** A transaction only sees data changes that were committed. Uncommitted changes (“dirty reads”) are not possible.
- **Repeatable Read Isolation Level:** A transaction can view changes made by the other transaction only after both transactions issue a COMMIT or both are rolled-back.
- **Serializable Isolation Level:** The strictest isolation level. Any concurrent execution of a set of serializable transactions is guaranteed to produce the same effect as running them sequentially in the same order.

Isolation levels affect the following database behavior:

- **Dirty Reads:** A transaction can read data that was written by another transaction, but is not yet committed.
- **Non-Repeatable (Fuzzy) Reads:** When reading the same data several times, a transaction can find that the data has been modified by another transaction that has just committed. The same query executed twice can return different values for the same rows.
- **Phantom Reads:** Similar to a non-repeatable read, but it is related to new data created by another transaction. The same query executed twice can return a different numbers of records.

Isolation level	Dirty read	Non-repeatable read	Phantom read
Read Uncommitted	Permitted	Permitted	Permitted
Read Committed	Not permitted	Permitted	Permitted
Repeatable Read	Not permitted	Not permitted	Permitted
Serializable	Not permitted	Not permitted	Not permitted

Oracle Isolation Levels

Oracle supports the Read Committed and Serializable isolation levels. It also provides a Read-Only isolation level which is not a part of the ANSI/ISO SQL standard (SQL92). Read Committed is the default.

- **Read Committed (default):** Each query executed within a transaction only sees data that was committed before the query itself. The Oracle database never allows reading “dirty pages” and uncommitted data.
- **Serializable:** Serializable transactions do not experience non-repeatable reads or phantom reads because they are only able to query changes that were committed at the time the transaction began (in addition to the changes made by the transaction itself performing DML operations).
- **Read-Only:** The read-only isolation level does not allow any DML operations during the transaction and only sees data committed at the time the transaction began.

Oracle Multiversion Concurrency Control (MVCC)

Oracle uses the MVCC mechanism to provide automatic read consistency across the entire database and all sessions. Using MVCC, database sessions query data based on a single point in time ensuring only committed changes are viewable. Oracle relies on the System Change Number (SCN) of the current transaction to obtain a consistent view of the database. Therefore, all database queries only return data committed with respect to the SCN at the time of query execution.

Setting Isolation Levels

Isolation levels can be changed at the transaction and session levels.

Examples

Change the isolation level at the transaction-level.

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;
```

Change the isolation-level at a session-level.

```
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
```

<https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/transactions.html#GUID-B97790CB-DF82-442D-B9D5-50CCE6BF9FBD>

MySQL Usage

Aurora MySQL supports all four transaction isolation levels described by the [SQL:1992 standard](#): READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE.

The default isolation level for Aurora MySQL is REPEATABLE READ.

The simplified syntax for setting transaction boundaries in Aurora MySQL is:

```
SET [SESSION] TRANSACTION ISOLATION LEVEL [READ WRITE | READ ONLY] | REPEATABLE READ |
READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE]
```

Note: Setting the GLOBAL isolation level is not supported in Aurora MySQL; only session scope can be changed. This behavior is similar to Oracle. Also, the default behavior of transactions is to use REPEATABLE READ and consistent reads. Applications designed to run with READ COMMITTED may need to be modified. Alternatively, explicitly change the default to READ COMMITTED.

To set the transaction isolation level, you will need to set the `tx_isolation` parameter (when using Aurora for MySQL), to see how to change parameters please follow the [parameters topic](#) in this book.

In Aurora MySQL, a Transaction Intent can be optionally specified. Setting a transaction to READ ONLY disables the transaction's ability to modify or lock both transactional and non-transactional tables visible to other transactions. The transaction can still modify or lock temporary tables. This enables internal optimization to improve performance and concurrency. The default is READ WRITE.

RDS MySQL Only (not Aurora MySQL): A new dynamic variable `innodb_deadlock_detect` may be used to disable deadlock detection in MySQL 8. On high concurrency systems deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times it may be more efficient to disable deadlock detection and rely on the `innodb_lock_wait_timeout` setting for transaction rollback when a deadlock occurs.

RDS MySQL Only (not Aurora MySQL): Starting MySQL8 InnoDB supports NOWAIT and SKIP LOCKED options with SELECT ... FOR SHARE and SELECT ... FOR UPDATE locking read statements. NOWAIT causes the statement to return immediately if a requested row is locked by another transaction. SKIP LOCKED removes locked rows from the result set. See Locking Read Concurrency with NOWAIT and SKIP LOCKED. SELECT ... FOR SHARE replaces SELECT ... LOCK IN SHARE MODE but LOCK IN SHARE MODE remains available for backward compatibility. The statements are equivalent. However FOR UPDATE and FOR SHARE support NOWAIT SKIP LOCKED and OF `tbl_name` options. See [Section 13.2.10](#).

Defining the Beginning of a Transaction

```
START TRANSACTION WITH CONSISTENT SNAPSHOT | READ WRITE | READ ONLY
```

Or

```
BEGIN [WORK]
```

The 'WITH CONSISTENT SNAPSHOT' option starts a consistent read Transaction. The effect is the same as issuing a START TRANSACTION followed by a SELECT from any table. WITH CONSISTENT SNAPSHOT does not change the transaction isolation level.

A consistent read uses snapshot information to make query results available based on a point-in-time regardless of modifications performed by concurrent transactions. If queried data has been changed by another transaction, the original data is reconstructed using the undo log. This avoids locking issues that may reduce concurrency. With the REPEATABLE READ isolation level, the snapshot is based on the time the first read operation is performed. With the READ COMMITTED isolation level, the snapshot is reset to the time of each consistent read operation.

Committing work at the end of a transaction

```
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
```

Rolling back work at the end of a transaction

```
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
```

One of the ROLLBACK options is ROLLBACK TO SAVEPOINT<logical_name>, this command will rollback all changes in current transaction up to the savepoint mentioned.

Create transaction savepoint during the transaction

```
SAVEPOINT <logical_name>
```

Note: If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The AND CHAIN clause causes a new transaction to begin as soon as the current one ends using the same isolation level and access mode as the just-terminated transaction.

The RELEASE clause causes the server to disconnect the current session after terminating the current transaction. The NO keyword suppresses both CHAIN and RELEASE completion. This can be useful if the completion_type system variable is set to cause chaining or release completion.

Always run with autocommit mode enabled. Set the autocommit parameter to 1 on the database side (which is the default) and on the application side (which might not be the default).

Always double-check the autocommit settings on the application side. For example, Python drivers such as MySQLdb and PyMySQL disable autocommit by default.

Aurora MySQL supports both auto commit and explicit commit modes. You can change the mode using the system variable 'autocommit', 1 is the default:

```
SET autocommit = {0 | 1}
```

Examples

Execute two DML statements within a serializable transaction.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
INSERT INTO Table1 VALUES (1, 'A');
```

```
UPDATE Table2 SET Column1 = 'Done' WHERE KeyColumn = 1;
COMMIT;
```


Summary

The following table summarizes the key differences in transaction support and syntax when migrating from Oracle to Aurora MySQL.

Transaction Property	Oracle	Aurora MySQL	Comments
Default isolation level	READ COMMITTED	REPEATABLE READ	The Aurora MySQL default isolation level is stricter than the Oracle. Evaluate application needs and set appropriately.
initialize transaction syntax	START TRANSACTION	START TRANSACTION	
Commit transaction	COMMIT [WORK FORCE]	COMMIT [WORK]	If using only COMMIT or COMMIT WORK, no change needed. Otherwise, rewrite FORCE to WORK.
Rollback transaction	ROLLBACK [WORK [TO FORCE]	ROLLBACK [WORK]	If using only ROLLBACK or ROLLBACK WORK, no change needed. Otherwise, rewrite TO and FORCE to WORK.
Set autocommit off/on	SET AUTOCOMMIT ON OFF (SQL*Plus)	SET autocommit = 0 1	
ANSI Isolations	REPEATABLE READ READ COMMITTED READ UNCOMMITTED SERIALIZABLE	REPEATABLE READ READ COMMITTED READ UNCOMMITTED SERIALIZABLE	Compatible syntax
MVCC	START TRANSACTION READ COMMITTED	WITH CONSISTENT SNAPSHOT	Aurora MySQL consistent read in READ COMMITTED isolation, is similar to Oracle's READ COMMITTED.
Nested transactions	Supported by starting new transaction or call a procedure/function after start transaction	Not Supported	Starting a new transaction in Aurora MySQL while another transaction is active causes a COMMIT of the previous transaction.
Transaction Chaining	Not Supported	Causes a new transaction to open immediately upon transaction completion	
Transaction Release	Not supported	Causes the client session to disconnect upon transaction completion	

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html>

Oracle Anonymous Block vs. MySQL Transactions or Procedures

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Different syntax may require code rewrite.

Oracle Usage

Oracle PL/SQL is a procedural extension of SQL. The PL/SQL program structure divides the code into blocks distinguished by the following keywords: DECLARE, BEGIN, EXCEPTION, and END.

An unnamed PL/SQL code block (code not stored in the database as a procedure, function, or package) is known as an anonymous block. An anonymous block serves as the basic unit of Oracle PL/SQL and contains the following code sections:

- **The Declarative Section (Optional):** Contains variables (names, data types, and initial values).
- **The Executable Section (Mandatory):** Contains executable statements (each block structure must contain at least one executable PL/SQL statement).
- **The Exception-Handling Section (Optional):** Contains elements for handling exceptions or errors in the code.

Examples

Simple structure of an Oracle Anonymous Block:

```
SET SERVEROUTPUT ON;
BEGIN
DBMS_OUTPUT.PUT_LINE('hello world');
END;
/

hello world
PL/SQL procedure successfully completed.
```

Oracle PL/SQL Anonymous blocks can contain advanced code elements such as functions, cursors, dynamic SQL, and conditional logic. The following anonymous block uses a cursor, conditional logic, and exception-handling.

```
SET SERVEROUTPUT ON;
DECLARE
v_sal_chk          NUMBER;
v_emp_work_years  NUMBER;
v_sql_cmd          VARCHAR2(2000);
BEGIN
FOR v IN (SELECT EMPLOYEE_ID, FIRST_NAME||' '||LAST_NAME AS
EMP_NAME, HIRE_DATE, SALARY FROM EMPLOYEES)
LOOP
v_emp_work_years:=EXTRACT(YEAR FROM SYSDATE) - EXTRACT (YEAR FROM v.hire_date);
```

```
IF v_emp_work_years>=10 and v.salary <= 6000 then
DBMS_OUTPUT.PUT_LINE('Consider a Bonus for: '||v.emp_name);
END IF;
END LOOP;
EXCEPTION WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('CODE ERR: '||sqlerrm);
END;
/
```

The above example calculates the number of years each employee has worked based on the HIRE_DATE column of the EMPLOYEES table. If the employee has worked for ten or more years and has a salary of \$6000 or less, the system prints the message “Consider a Bonus for: <employee name>”.



For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/overview.html#GUID-2FBCFBBE-6B42-4DB8-83F3-55B63B75B1EB>

MySQL Usage

Functionality similar to Oracle Anonymous Blocks can be achieved the using the Aurora MySQL START TRANSACTION command or a stored procedure.

For more information, see [Stored Procedures](#) or [Transactions](#).

Oracle Conversion functions vs. MySQL Conversion Functions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Not all functions are supported by MySQL and may require to create manually

Oracle Usage

All databases have their own conversion methods for transforming data between types and performing data manipulation. This section addresses the conversion functions TO_CHAR and TO_NUMBER.

TO_CHAR

TO_CHAR can convert many types of data (mostly number, date, and string) to string. There are many format combinations. Some examples include:

To_char calls with strings	Results
to_char('0972')	0972
to_char('0972','9999')	972
to_char('0972','\$9999.99')	\$972.00
to_char('0972','\$0009999.99')	\$972.00
to_char('0972.48','\$9999.999')	\$972.480

To_char calls with numbers	Results
to_char(0972)	972
to_char(0972,'9999')	972
to_char(0972,\$9999.99')	\$972.00
to_char(0972,\$0009999.99')	\$0000972.00
to_char(0972.48,\$9999.999')	\$972.480

To_char calls with date	Results	description
to_char(sysdate,'YYYY')	2013	Year
to_char(sysdate,'YY')	13	Last two digits of the year
to_char(sysdate,'YEAR')	TWENTY THIRTEEN	Year in words
to_char(sysdate,'SYYYY')	2017	S prefixed (-) sign for BC

To_char calls with date	Results	description
to_char(sysdate,'Y,YYYY')	2017	Year with comma
to_char(sysdate,'MONTH')	SEPTEMBER	Complete month
to_char(sysdate,'MON')	SEP	3 letter month format
to_char(sysdate,'MM')	9	Month of the year
to_char(sysdate,'W')	4	Week of the current month
to_char(sysdate,'WW')	36	Week of the year (1 - 53)
to_char(sysdate,'DAY')	SATURDAY	Name of the day
to_char(sysdate,'DD')	30	Day in number format
to_char(sysdate,'D')	7	Day of the week (1 - 7)
to_char(sysdate,'DDD')	273	Day of the year (1 - 366)
to_char(sysdate,'DY')	SAT	Short form of Day
to_char(sysdate,'HH')	9	Hour (1 - 12)
to_char(sysdate,'HH12')	9	Hour in 12 hours format
to_char(sysdate,'HH24')	21	Hour in 24 hours format
to_char(sysdate,'MI')	15	Minute (0 - 59)
to_char(sysdate,'SS')	24	Second (0 - 59)
to_char(sysdate,'SSSS')	79100	Seconds after midnight (0 - 86399)
to_char(sysdate,'PM')	PM	AM or PM
to_char(sysdate,'AM')	PM	AM or PM
to_char(sysdate,'DL')	Saturday, February 23, 2017	Date in Long Format
to_char(sysdate,'Q')	3	Quarter of the Year (1 - 4)

TO_NUMBER

TO_NUMBER converts one of the following to number data types: CHAR, VARCHAR2, NCHAR, NVARCHAR2, BINARY_FLOAT, or BINARY_DOUBLE. When converting one of the first four types, you can use the format parameter for the returned number.

The format parameter specifies one of the following options:

Example data to convert	Format parameter	Results
-1234567890	9999999999S	'1234567890-'
0	99.99	' .00'
0.1	99.99	' .10'
-0.2	99.99	' -.20'
0	9999	' 0'
1	9999	' 1'
0	B9999	''

Example data to convert	Format parameter	Results
1	B9999	'1'
123.456	999.999	'123.456'
123.456	FM999.009	'123.456'
123.456	9.9E0000	'1.2E+02'
1.00E+123	9.9E0000	'1.0E+123'
123.456	FM9.9E0000	'1.2E+02'
123.45	FM999.009	'123.45'
123	FM999.009	'123.00'
123.45	L999.99	'\$123.45'
123.45	FML999.99	'\$123.45'
1234567890	9999999999S	'1234567890+'

Examples

Convert a string to a number.

```
select to_number('99999') from dual;
```

```
TO_NUMBER('99999')
```



```
-----  
99999
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Functions.html#GUID-D079EFD3-C683-441F-977E-2C9503089982>

MySQL Usage

See [Single-Row and Aggregate Functions](#).

Oracle PL/SQL Cursors vs. MySQL Cursors

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Cursors	<p>Minor different in syntax may require some code rewrite.</p> <p>%ISOPEN, %ROWTYPE, and %BULK_ROWCOUNT are not supported</p>

Oracle Usage

PL/SQL cursors are pointers to data sets on which application logic can iterate. The data sets hold rows returned by SQL statements. You can refer to the active data set in named cursors from within a program.

There are two types of PL/SQL cursors :

- **Implicit Cursors** are session cursors constructed and managed by PL/SQL automatically without being created or defined by a user. PL/SQL opens an implicit cursor each time you run a SELECT or DML statement. Implicit cursors are also called SQL Cursors.
- **Explicit Cursors** are session cursors created, constructed, and managed by a user. Cursors are declared and defined by naming it and associating it with a query. Unlike an implicit cursor, you can reference an explicit cursor using its name. An explicit cursor is called a Named Cursor.

Examples

The following examples demonstrate cursor usage:

First example:

1. Define an explicit PL/SQL cursor named c1.
2. The cursor executes an SQL statement to return rows from the database.
3. The PL/SQL Loop reads data from the cursor, row by row, and stores the values into two variables: v_lastname and v_jobid.
4. The loop uses the %NOTFOUND attribute to terminate when the last row is read from the database.

```

DECLARE
  CURSOR c1 IS
    SELECT last_name, job_id FROM employees
    WHERE REGEXP_LIKE (job_id, 'S[HT]_CLERK')
    ORDER BY last_name;
  v_lastname employees.last_name%TYPE; -- variable to store last_name
  v_jobid employees.job_id%TYPE; -- variable to store job_id
BEGIN
  OPEN c1;
  LOOP -- Fetches 2 columns into variables
    FETCH c1 INTO v_lastname, v_jobid;
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
  CLOSE c1;
END;
```

Second example:

1. Define an implicit PL/SQL Cursor using a FOR Loop.
2. The cursor executes a query and stores values returned into a record.
3. A loop iterates over the Cursor data set and prints the result.

```
BEGIN
FOR item IN
  (SELECT last_name, job_id FROM employees WHERE job_id LIKE      '%MANAGER%' AND
manager_id > 400 ORDER BY last_name) LOOP
  DBMS_OUTPUT.PUT_LINE('Name = ' || item.last_name || ', Job = ' ||
                        item.job_id);
END LOOP;
END;
/
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/explicit-cursor-declaration-and-definition.html#GUID-38C5DBA3-9DEC-4AF2-9B5E-7B721D11A77C>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/implicit-cursor-attribute.html#GUID-5A938EE7-E8D2-468C-B60F-81898F110BE1>

MySQL Usage

Aurora MySQL supports cursors only within stored routines, functions and stored procedures.

Unlike Oracle, which offers an array of cursor types, Aurora MySQL Cursors have the following characteristics:

- **Asensitive:** The server can choose to either make a copy of its result table or to access the source data as the cursor progresses.
- **Read only:** Cursors cannot be updated.
- **Nonscrollable:** Cursors can only be traversed in one direction and cannot skip rows. The only supported cursor advance operation is FETCH NEXT.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Similar to Oracle, cursors are declared with the DECLARE CURSOR, opened with OPEN, fetched with FETCH, and closed with CLOSE.

DECLARE Cursor

```
DECLARE <Cursor Name> CURSOR FOR <Cursor SELECT Statement>
```

The DECLARE CURSOR statement instantiates a cursor object and associates it with a SELECT statement. This SELECT is then used to retrieve the cursor rows.

To fetch the rows, use the FETCH statement. As mentioned above, only FETCH NEXT is supported. The number of output variables specified in the FETCH statement must match the number of columns retrieved by the cursor.

Aurora MySQL cursors have additional characteristics:

- SELECT INTO is not allowed in a cursor.
- Stored routines can have multiple cursor declarations, but all cursors declared in a given code block must have a unique name.
- Cursors can be nested.

OPEN Cursor

```
OPEN <Cursor Name>;
```

The OPEN command populates the cursor with the data, either dynamically or in a temporary table, and readies the first row for consumption by the FETCH statement.

FETCH Cursor

```
FETCH [[NEXT] FROM] <Cursor Name> INTO <Variable 1> [,<Variable n>]
```

The FETCH statement retrieves the current pointer row, assigns the column values to the variables listed in the FETCH statement, and advances the cursor pointer by one row. If the row is not available, meaning the cursor has been exhausted, a No Data condition is raised with an SQLSTATE value of '0200000'. To catch this condition, or the alternative NOT FOUND condition, you must create a condition handler.

Note: Carefully plan your error handling flow. The same condition might be raised by SELECT statements or cursors other than the one you intended. Place operations within BEGIN ... END blocks to associate each cursor with its own handler.

CLOSE Cursor

```
CLOSE <Cursor Name>;
```

The CLOSE statement closes an open cursor. If the cursor with the specified name does not exist, an error is raised. If a cursor is not explicitly closed, Aurora MySQL closes it automatically at the end of the BEGIN ... END block in which it was declared.

Examples

The following example uses a cursor to iterate over source rows and merges into an OrderItems table.

Create an OrderItems table.

```
CREATE TABLE OrderItems(OrderID INT NOT NULL, Item VARCHAR(20) NOT NULL, Quantity SMALLINT NOT NULL, PRIMARY KEY(OrderID, Item));
```

Create and populate the SourceTable.

```
CREATE TABLE SourceTable (OrderID INT, Item VARCHAR(20), Quantity SMALLINT, PRIMARY KEY (OrderID, Item));

INSERT INTO SourceTable (OrderID, Item, Quantity) VALUES (1, 'M8 Bolt', 100), (2, 'M8 Nut', 100), (3, 'M8 Washer', 200);
```

Create a procedure to loop through SourceTable and insert rows.

```

CREATE PROCEDURE LoopItems ()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE var_OrderID INT;
  DECLARE var_Item VARCHAR(20);
  DECLARE var_Quantity SMALLINT;
  DECLARE ItemCursor CURSOR FOR SELECT OrderID, Item, Quantity FROM SourceTable;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN ItemCursor;
  CursorStart: LOOP
  FETCH NEXT FROM ItemCursor INTO var_OrderID, var_Item, var_Quantity;
  IF Done THEN LEAVE CursorStart;
  END IF;
  INSERT INTO OrderItems (OrderID, Item, Quantity) VALUES (var_OrderID, var_Item,
var_Quantity);
  END LOOP;
  CLOSE ItemCursor;
END;

```

Execute the stored procedure.

```
CALL LoopItems ();
```

Select all rows from the OrderItems table.

```
SELECT * FROM OrderItems;
```

OrderID	Item	Quantity
-----	----	-----
1	M8 Bolt	100
2	M8 Nut	100
3	M8 Washer	200

Summary



The following table describes similarities, differences, and key migration considerations.

Action	Oracle	Aurora MySQL
Declare a bound explicit cursor	<code>CURSOR c1 IS SELECT * FROM employees;</code>	<code>DECLARE c1 CURSOR FOR SELECT * FROM employees;</code>
Open a cursor	<code>OPEN c1;</code>	<code>OPEN c1;</code>
Move Cursor to next row and fetch into a record variable (rowvar was declared in the DECLARE section)	<code>FETCH c1 INTO rowvar;</code>	<code>FETCH NEXT FROM c1 INTO rowvar;</code>
Move Cursor to next row and fetch into multiple scalar data types (emp_id, emp_name, salary was declared in the DECLARE section)	<code>FETCH c1 INTO emp_id, emp_name, salary;</code>	<code>FETCH NEXT FROM c1 INTO emp_id, emp_name, salary;</code>
Iterate through an Implicit Cursor via a Loop	<code>FOR item IN (SELECT last_name, job_id FROM employees WHERE job_id LIKE '%CLERK%' AND manager_id > 120 ORDER BY last_name) LOOP << do something >> END LOOP;</code>	N/A
Declare a cursor with variables	<code>CURSOR c1 (key NUMBER) IS SELECT * FROM employees WHERE id = key;</code>	<code>SET @sqltext1 := CONCAT ('DECLARE c1 CURSOR FOR SELECT * FROM employees WHERE id =',key); PREPARE stmt1 FROM @sqltext1; EXECUTE stmt1;</code>
Open a cursor with variables	<code>OPEN c1(2);</code>	Use regular OPEN after declaring the CURSOR using EXECUTE and PREPARE with variables
Exit a loop after no data found	<code>EXIT WHEN c1%NOTFOUND;</code>	<code>DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; And in the fetching loop insert: IF done THEN LEAVE;</code>
Detect if a Cursor has rows remaining in its dataset	<code>%FOUND</code>	N/A
Determine how many rows were affected from any DML statement	<code>%BULK_ROWCOUNT</code>	Use counters
Determine which DML execution failed with the relevant error code	<code>%BULK_EXCEPTIONS</code>	N/A
Detect if the Cursor is open	<code>%ISOPEN</code>	N/A

Action	Oracle	Aurora MySQL
Detect if a Cursor has no rows remaining in its dataset	%NOTFOUND	N/A
Returns the number of rows affected by a Cursor	%ROWCOUNT	N/A

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/cursors.html>

Oracle DBMS_DATAPUMP vs. MySQL Integration with Amazon S3

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	No equivalent tool

Oracle Usage

The DBMS_DATAPUMP package provides Oracle Data Pump functionality that can be run within the database.

The DBMS_DATAPUMP package subprograms are:

- ADD_FILE: Adds a relevant file to the dump file set.
- ATTACH: Connects the DATAPUMP job.
- DATA_FILTER: Filters rows.
- DETACH: Disconnects from a DATAPUMP operation.
- GET_DUMPFILE_INFO: Retrieves information about a specified dump file.
- GET_STATUS: Retrieves status of the running DATAPUMP operation.
- LOG_ENTRY: Writes a message into the log file.
- METADATA_FILTER: Filters the items to be include in the operation.
- METADATA_REMAP: Remaps the object to new names.
- METADATA_TRANSFORM: Specifies transformations to be applied to objects.
- OPEN: Declares a new job.
- SET_PARALLEL: Set the parallelism of the job.
- SET_PARAMETER: Specifies job processing options.
- START_JOB: Executes a job.
- STOP_JOB: Terminates a job.
- WAIT_FOR_JOB: Runs a job until it either completes normally or stops.

Examples

The following example demonstrates exporting the HR schema. It assumes all directories have already been created and the user has all required privileges.

```
DECLARE
loopidx NUMBER;
job_handle NUMBER;
percent_done NUMBER;
job_state VARCHAR2(30);
err ku$_LogEntry;
job_status ku$_JobStatus;
job_desc ku$_JobDesc;
```

```

obj_stat ku$_Status;
BEGIN

job_handle := DBMS_DATAPUMP.OPEN('EXPORT','SCHEMA',NULL,'EXP_SAMP','LATEST');

DBMS_DATAPUMP.ADD_FILE(job_handle,'hr.dmp','DMPDIR');

DBMS_DATAPUMP.METADATA_FILTER(job_handle,'SCHEMA_EXPR','IN (''HR'')');

DBMS_DATAPUMP.START_JOB(job_handle);

percent_done := 0;
job_state := 'UNDEFINED';
while (job_state != 'COMPLETED') and (job_state != 'STOPPED') loop
dbms_datapump.get_status(job_handle,
dbms_datapump.ku$_status_job_error +
dbms_datapump.ku$_status_job_status +
dbms_datapump.ku$_status_wip,-1,job_state,obj_stat);
job_status := obj_stat.job_status;

/* HERE YOU CAN PRINT THE STATUS */

if job_status.percent_done != percent_done then
    percent_done := job_status.percent_done;
end if;

if (bitand(obj_stat.mask,dbms_datapump.ku$_status_wip) != 0) then
    err := obj_stat.wip;
else
    if (bitand(obj_stat.mask,dbms_datapump.ku$_status_job_error) != 0)
    then
        err := obj_stat.error;
    else
        err := null;
    end if;
end if;

if err is not null then
    loopidx := err.FIRST;
    while loopidx is not null loop
        loopidx := err.NEXT(loopidx);
    end loop;
end if;
end loop;

dbms_datapump.detach(job_handle);
END;
/

```

For more details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sutil/oracle-data-pump-overview.html#GUID-17FAE261-0972-4220-A2E4-44D479F519D4>

MySQL Usage

There is no feature in MySQL fully equivalent to the Oracle DBMS_DATAPUMP package, but there are tools and features that achieve the same functionality.

To export data from the database to the file system, use the SELECT INTO OUTFILE S3 command. To import data from the filesystem, use the LOAD DATA FROM S3 command.

To achieve the most functionality, this feature can be mixed with metadata tables and events to handle the operations.



For additional details, see: For additional information, see: [S3 integration](#)

Summary

Feature	Oracle DBMS_DATAPUMP	Aurora integration with S3
Add a relevant file to the dump file set	ADD_FILE	Use metadata table
Connect the DATAPUMP job	ATTACH	Query session status
Filter rows to be handled	DATA_FILTER	Use WHERE clause in your SELECT
Disconnect from DATAPUMP operation	DETACH	Not required
Retrieve information about a specified dump file	GET_DUMPFILINFO	Use metadata table
Retrieve the status of the running DATAPUMP operation	GET_STATUS	Query session status
Write a message into the log file	LOG_ENTRY	Write to metadata tables
Filter the items included in the operation	METADATA_FILTER	Use export the objects
Remap the object to new names	METADATA_REMAP	LOAD DATA INTO different table name
Specified transformations to be applied to objects	METADATA_TRANSFORM	Not required
Declare a new job	OPEN	Use LOAD DATA or SAVE OUTFILE
Set the parallelism of the job.	SET_PARALLEL	Use parallel in your SELECT
Specify job-processing options	SET_PARAMETER	Not required
Execute a job	START_JOB	Use LOAD DATA or SAVE OUTFILE
Terminates a job	STOP_JOB	Kill session

Feature	Oracle DBMS_DATAPUMP	Aurora integration with S3
Runs a job until it either completes normally or stops	WAIT_FOR_JOB	Use LOAD DATA or SAVE OUTFILE

Oracle DBMS_OUTPUT vs. MySQL SELECT

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		DBMS_OUTPUT	Different paradigm and syntax requires application/drivers rewrite

Oracle Usage

The Oracle DBMS_OUTPUT package is typically used for debugging or for displaying output messages from PL/SQL procedures.

Examples

In the following example, DBMS_OUTPUT with PUT_LINE is used with a combination of bind variables to dynamically construct a string and print a notification to the screen from within an Oracle PL/SQL procedure. To display notifications on the screen, you must configure the session with SET SERVEROUTPUT ON.

```
SET SERVEROUTPUT ON
DECLARE
CURSOR c1 IS
SELECT last_name, job_id FROM employees
WHERE REGEXP_LIKE (job_id, 'S[HT]_CLERK')
ORDER BY last_name;
v_lastname employees.last_name%TYPE; -- variable to store last_name
v_jobid employees.job_id%TYPE; -- variable to store job_id
BEGIN
OPEN c1;
LOOP -- Fetches 2 columns into variables
FETCH c1 INTO v_lastname, v_jobid;
DBMS_OUTPUT.PUT_LINE ('The employee id is:' || v_jobid || ' and his last name is:' ||
v_lastname);
EXIT WHEN c1%NOTFOUND;
END LOOP;
CLOSE c1;
END;
```

In addition to the output of information on the screen, the PUT and PUT_LINE procedures in the DBMS_OUTPUT package enable you to place information in a buffer that can be read later by another PL/SQL procedure or package. You can display the previously buffered information using the GET_LINE and GET_LINES procedures.

For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_OUTPUT.html#GUID-C1400094-18D5-4F36-A2C9-D28B0E12FD8C

MySQL Usage

In Aurora MySQL, SELECT can be used to display output messages.

Examples

```

delimiter //

CREATE PROCEDURE emp_counter (param1 INTEGER)
BEGIN
SELECT "" 'OUTPUT: Before count';
SELECT COUNT(*) INTO param1 FROM EMPS;
SELECT concat('Employees count: ', param1) as '';
SELECT "" 'OUTPUT: After count';
END//



delimiter ;
call simpleprocl(1);
+-----+
|           |
+-----+
| OUTPUT: Before count |
+-----+
1 row in set (0.19 sec)
+-----+
|           |
+-----+
| Employees count: 8 |
+-----+
1 row in set (0.20 sec)
+-----+
|           |
+-----+
| OUTPUT: After count |
+-----+
1 row in set (0.21 sec)
Query OK, 0 rows affected (0.22 sec)

```

Note: Use quotes (") with SELECT for cleaner display. Otherwise, messages are displayed twice, both as header and value.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/select.html>

Oracle DBMS_RANDOM vs. MySQL RAND Function

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Different syntax and missing options may require code rewrite

Oracle Usage

The Oracle DBMS_RANDOM package provides functionality for generating random numbers or strings as part of an SQL statement or PL/SQL procedure.

The DBMS_RANDOM Package stored procedures include:

- **NORMAL**: Returns random numbers in a standard normal distribution.
- **SEED**: Resets the seed that generates random numbers or strings.
- **STRING** Returns a random string.
- **VALUE**: Returns a number greater than or equal to 0 and less than 1 with 38 digits to the right of the decimal. Alternatively, you could generate a random number greater than or equal to a low parameter and less than a high parameter.

Notes

- DBMS_RANDOM.RANDOM produces integers in the range $[-2^{31}, 2^{31}]$.
- DBMS_RANDOM.VALUE produces numbers in the range $[0, 1]$ with 38 digits of precision.

Examples

Generate a random number.

```
select dbms_random.value() from dual;

DBMS_RANDOM.VALUE ()
-----
.859251508

select dbms_random.value() from dual;

DBMS_RANDOM.VALUE ()
-----
.364792387
```

Generate a random string. The first character determines the returned string type and the number specifies the length.

```
select dbms_random.string('p',10) from dual;
DBMS_RANDOM.STRING('P',10)
```

```

-----
la'?'z[Q&/2

select dbms_random.string('p',10) from dual;
DBMS_RANDOM.STRING('P',10)
-----
t?!Gf2M60q

```

For more details, see: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_RANDOM.htm-#GUID-8DC48B0C-3707-4172-A306-C0308DD2EB0F

MySQL Usage

The MySQL RAND function is not fully equivalent to Oracle DBMS_RANDOM because it does not generate string values. However, there are other functions in that can be used in combination to achieve full functionality.

RAND function returns a random floating-point value v in the range $0 \leq v < 1.0$.

The RAND function can be used with a seed value to reset the seed. If an integer argument N is specified, it is used as the seed value:

- With a constant initializer argument, the seed is initialized once when the statement is prepared and prior to execution.
- With a non-constant initializer argument (such as a column name), the seed is initialized with the value for each invocation of RAND().

Examples

Generate a random number.

```

select RAND();

RAND() |
-----|
0.30244802525494996 |

```

To obtain a random integer R in the range $i \leq R < j$, use the expression $\text{FLOOR}(i + \text{RAND}() * (j - i))$. For example, to obtain a random integer in the range $7 \leq R < 12$, use:

```

SELECT FLOOR(7 + (RAND() * 5));

FLOOR(7 + (RAND() * 5)) |
-----|
8 |

```

Generate an eight-character string of digits.

```

SELECT SUBSTRING(MD5(RAND())) FROM 1 FOR 8);



```

Generate an eight-character string containing characters only.

```
SELECT concat(substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand
()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1),
substring('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', rand()*52+1, 1))
```

For more details, see https://dev.mysql.com/doc/refman/5.7/en/mathematical-functions.html#function_rand

Oracle DBMS_REDEFINITION vs. MySQL Tables and Triggers

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL does not support DBMS_REDEFINITION

Oracle Usage

The Oracle DBMS_REDEFINITION package can be used to reorganize tables while they perform DML operations. Use this package to reclaim space due to a high watermark or to change the table's DDL.

Oracle uses materialized views to track changes on the master table and then applies those changes in refresh synchronization.

Examples

Run online redefinition.

- DBMS_REDEFINITION.CAN_REDEF_TABLE: Determines if the table can be redefined.
- DBMS_REDEFINITION.START_REDEF_TABLE: Start the online redefinition.
- DBMS_REDEFINITION.SYNC_INTERIM_TABLE: Synchronize tables with interim data.
- DBMS_REDEFINITION.FINISH_REDEF_TABLE: Complete redefinition.

```
EXEC DBMS_REDEFINITION.CAN_REDEF_TABLE('HR', 'EMPLOYEES');
CREATE TABLE employees2 AS SELECT * FROM employees WHERE 1=2;

EXEC DBMS_REDEFINITION.START_REDEF_TABLE
    ('HR', 'EMPLOYEES', 'EMPLOYEES2', '*');

EXEC DBMS_REDEFINITION.SYNC_INTERIM_TABLE
    ('HR', 'EMPLOYEES', 'EMPLOYEES2');

ALTER TABLE employees2 ADD
    (CONSTRAINT emp_pk2 PRIMARY KEY (empno) USING INDEX);

EXEC DBMS_REDEFINITION.FINISH_REDEF_TABLE
    ('HR', 'EMPLOYEES', 'EMPLOYEES2');

DROP TABLE employees2;
```

For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_REDEFINITION.html#GUID-2BA796C4-8B4D-49B4-8A35-4C6F789CD374

MySQL Usage



MySQL has no equivalent to Oracle for automatically rebuilding tables or syncing between two tables. However, you can sync data from one table to another using CREATE TABLE AS SELECT or mysqldump. After the table is copied, the delta rows can be copied using triggers. Once the application is ready to use the new table, it is synced.

If a table has sequence columns, the last value in the sequence is retained when the table is copied.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-table-select.html>
- <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>

Oracle DBMS_SQL Package (11g & 12c) vs. MySQL DBMS_SQL Package

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Different paradigm and syntax requires application/drivers rewrite.

Oracle Usage

The DBMS_SQL package provides an interface to parse and execute dynamic SQL statements, DML commands, and DDL commands (usually from within a PL/SQL package, function, or procedure). DBMS_SQL enables granular control of SQL cursors and can improve cursor performance in certain cases.

Examples

The following examples demonstrates how to manually open, parse, bind, execute, and fetch data from a cursor using the DBMS_SQL PL/SQL interface.

1. Use DBMS_SQL.OPEN_CURSOR to open a blank cursor and return the cursor handle.
2. Use DBMS_SQL.PARSE to parse the statement into the referenced cursor.
3. Use DBMS_SQL.BIND_VARIABLES to attach the value for the bind variable with the cursor.
4. Use DBMS_SQL.EXECUTE to execute the cursor.
5. Use DBMS_SQL.GET_NEXT_RESULT to iterate over the cursor, fetching the next result.
6. Use DBMS_SQL.CLOSE_CURSOR to close the cursor.

```

DECLARE
c1          INTEGER;
rc1         SYS_REFCURSOR;
n           NUMBER;
first_name  VARCHAR2(50);
last_name   VARCHAR2(50);
email       VARCHAR2(50);
phone_number VARCHAR2(50);
job_title   VARCHAR2(50);
start_date  DATE;
end_date    DATE;
BEGIN
c1 := DBMS_SQL.OPEN_CURSOR(true);
DBMS_SQL.PARSE
    (c1, 'BEGIN emp_info(:id); END;', DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE(c1, ':id', 176);
n := DBMS_SQL.EXECUTE(c1);

-- Get employee info
DBMS_SQL.GET_NEXT_RESULT(c1, rc1);
FETCH rc1 INTO first_name, last_name, email, phone_number;

```

```

-- Get employee job history
DBMS_SQL.GET_NEXT_RESULT(c1, rc1);
LOOP
FETCH rc1 INTO job_title, start_date, end_date;
EXIT WHEN rc1%NOTFOUND;
END LOOP;
DBMS_SQL.CLOSE_CURSOR(c1);
END;
/

```

There are three other procedures in DBMS_SQL package:

- **RETURN_RESULT:** (New in oracle 12c) Gets a result set and returns it to the client. Because the procedure already returns a result set, the invoker does not have to know the format of the result or the columns it contains (most often used with SQL*Plus).
- **TO_REFCURSOR:** When using DBMS_SQL.OPEN_CURSOR, the numeric cursor ID is returned. If you know the structure of the result of the cursor, you can call the TO_REFCURSOR procedure, stop working with DBMS_SQL, and move to regular commands such as FETCH, WHEN CURSOR%notfound, and others. Before using TO_REFCURSOR, use the procedures OPEN_CURSOR, PARSE and EXECUTE.
- **TO_CURSOR_NUMBER:** Gets a cursor opened in native dynamic SQL. After the cursor is open, it can be converted to a number (cursor id) and then managed using DBMS_SQL procedures.

For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_SQL.html#GUID-C96D5BAA-29A9-4AB5-A69E-E31228ECC9E9

MySQL Usage



There is no equivalent feature for the DBMS_SQL package. The only options for Aurora MySQL are:

- Procedures or Functions
- Prepare and Execute statements

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-procedure.html>
- <https://dev.mysql.com/doc/refman/5.7/en/execute.html>
- <https://dev.mysql.com/doc/refman/5.7/en/prepare.html>
- <https://dev.mysql.com/doc/refman/5.7/en/sql-syntax-prepared-statements.html>

Oracle EXECUTE IMMEDIATE vs. MySQL Execute and Prepare

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		EXECUTE IMMEDIATE	<p>Must use PREPARE command in MySQL</p> <p>Execute SQL with results and bind variables or Anonymous block execution using EXECUTE are not supported in MySQL</p>

Oracle Usage

The Oracle EXECUTE IMMEDIATE statement can be used to parse and execute a dynamic SQL statement or an anonymous PL/SQL block. It also supports bind variables.

Examples

Run a dynamic SQL statement from within a PL/SQL procedure:

1. Create a PL/SQL procedure named raise_sal.
2. Define a SQL Statement with a dynamic value for the column name included in the where statement.
3. Use the EXECUTE IMMEDIATE command supplying the two bind variables to be used as part of the SELECT statement: amount and col_val.

```
CREATE OR REPLACE PROCEDURE raise_sal (col_val NUMBER,
emp_col VARCHAR2, amount NUMBER) IS
  col_name VARCHAR2(30);
  sql_stmt   VARCHAR2(350);
BEGIN
  -- determine if a valid column name has been given as input
  SELECT COLUMN_NAME INTO col_name FROM USER_TAB_COLS
  WHERE TABLE_NAME = 'EMPLOYEES' AND COLUMN_NAME = emp_col;

  -- define the SQL statment (with bind variables)
  sql_stmt := 'UPDATE employees SET salary = salary + :1 WHERE ' ||
  col_name || ' = :2';

  -- Execute the command
  EXECUTE IMMEDIATE sql_stmt USING amount, col_val;
END raise_sal;
/
```

4. Run the DDL operation from within an EXECUTE IMMEDIATE command.

```
EXECUTE IMMEDIATE 'CREATE TABLE link_emp (idemp1 NUMBER, idemp2 NUMBER)';
EXECUTE IMMEDIATE 'ALTER SESSION SET SQL_TRACE TRUE';
```

5. Run an anonymous block with bind variables using EXECUTE IMMEDIATE.

```
EXECUTE IMMEDIATE 'BEGIN raise_sal (:col_val, :col_name, :amount); END;'
USING 134, 'EMPLOYEE_ID', 10;
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/Inpls/EXECUTE-IMMEDIATE-statement.html#GUID-C3245A95-B85B-4280-A01F-12307B108DC8>

MySQL Usage

MySQL Execute

The EXECUTE command executes commands that were prepared by the PREPARE command. It can also run DDL statements and retrieve data using SQL commands. Similar to Oracle, the MySQL EXECUTE command can be used with bind variables.

MySQL Prepare

The PREPARE command can receive a SELECT, INSERT, UPDATE, DELETE, or VALUES statement and parse it with a user-specified qualifying name so that the EXECUTE command can be used later without the need to re-parse the SQL statement for each execution.

- Statement names are not case-sensitive. A Statement name is either a string literal or a user variable containing the text of the SQL statement.
- If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared.
- The scope of a prepared statement is the session in which it is created.

Examples

Execute an SQL SELECT query with the table name as a dynamic variable using bind variables. This query returns the number of employees under a manager with a specific ID.

```
PREPARE stmt1 FROM 'SELECT count(*) FROM employees WHERE ID=?';
SET @man_id = 3;
EXECUTE stmt1 USING @a;
```

```
count(*) |
-----|
2         |
```

Execute a DML command—first with no variables and then with variables.

```
PREPARE stmt1 FROM 'INSERT INTO numbers (a) VALUES (1)';
EXECUTE stmt1;
```

```
PREPARE stmt1 FROM 'INSERT INTO numbers (a) VALUES (?)';
SET @man_id = 3;
EXECUTE stmt1 USING @a;
```

Execute a DDL command.

```
PREPARE stmt1 FROM 'CREATE TABLE numbers (num integer)';
EXECUTE stmt1;
```



Summary

Functionality	Oracle - EXECUTE IMMEDIATE	MySQL - EXECUTE & PREPARE
Execute SQL with results and bind variables	EXECUTE IMMEDIATE 'select salary from employees WHERE ' col_name ' = :1' INTO amount USING col_val;	N/A
Execute DML with variables and bind variables	EXECUTE IMMEDIATE 'UPDATE employees SET salary = salary + :1 WHERE ' col_name ' = :2' USING amount, col_val;	PREPARE stmt1 FROM 'UPDATE employees SET salary = salary + ? WHERE ? = ?' EXECUTE stmt1 USING @amount, @col, @colval;
Execute DDL	EXECUTE IMMEDIATE 'CREATE TABLE link_emp (idemp1 NUMBER, idemp2 NUMBER)';	PREPARE stmt1 FROM 'CREATE TABLE link_emp (idemp1 INTEGER, idemp2 INTEGER)' EXECUTE stmt1;
Execute Anonymous block	EXECUTE IMMEDIATE 'BEGIN DBMS_OUTPUT.PUT_LINE ("Anonymous Block"); END;';	N/A

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/execute.html>
- <https://dev.mysql.com/doc/refman/5.7/en/prepare.html>
- <https://dev.mysql.com/doc/refman/5.7/en/sql-syntax-prepared-statements.html>

Oracle Procedures and Functions vs. MySQL Stored Procedures

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Procedures	Syntax and option differences

Oracle Usage

Oracle PL/SQL is the built-in database programming language that provides several methods to store and execute reusable business logic from within the database. Procedures and functions are reusable snippets of code created using the CREATE PROCEDURE and the CREATE FUNCTION statements.

Stored Procedures and Stored Functions are PL/SQL units of code consisting of SQL and PL/SQL statements that solve specific problems or perform a set of related tasks.

- **Procedure:** Used to perform database actions with PL/SQL.
- **Function:** Used to perform a calculation and return a result.

Privileges for Creating Procedures and Functions

- To create procedures and functions in their own schema, Oracle database users must have the CREATE PROCEDURE system privilege.
- To create procedures or functions in other schemas, database users must have the CREATE ANY PROCEDURE privilege.
- To execute a procedure or function, database users must have the EXECUTE privilege.

Package and Package Body

In addition to stored procedures and functions, Oracle also provides Packages to encapsulate related procedures, functions, and other program objects.

- **Package:** Declares and describes all the related PL/SQL elements.
- **Package Body:** Contains the executable code.

To execute a stored procedure or function created inside a package, the package name and the stored procedure or function name must be specified.

```
EXEC PKG_EMP.CALCULATE_SAL('100');
```

Examples

Create an Oracle stored procedure using the CREATE OR REPLACE PROCEDURE statement. The optional OR REPLACE clause overwrites an existing stored procedure with the same name if it exists.

```
CREATE OR REPLACE PROCEDURE EMP_SAL_RAISE
(P_EMP_ID IN NUMBER, SAL_RAISE IN NUMBER)
```

```

AS
V_EMP_CURRENT_SAL NUMBER;
BEGIN
SELECT SALARY INTO V_EMP_CURRENT_SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=P_EMP_ID;
UPDATE EMPLOYEES
SET SALARY=V_EMP_CURRENT_SAL+SAL_RAISE
WHERE EMPLOYEE_ID=P_EMP_ID;
DBMS_OUTPUT.PUT_LINE('New Salary For Employee ID: '||P_EMP_ID||' Is '||(V_EMP_CURRENT_
SAL+SAL_RAISE));
EXCEPTION WHEN OTHERS THEN
RAISE_APPLICATION_ERROR(-20001,'An error was encountered - '||SQLCODE||' -ERROR-
'||SQLERRM);
ROLLBACK;
COMMIT;
END;
/
-- Execute
EXEC EMP_SAL_RAISE(200, 1000);

```

Create a function using the CREATE OR REPLACE FUNCTION statement.

```

CREATE OR REPLACE FUNCTION EMP_PERIOD_OF_SERVICE_YEAR
(P_EMP_ID NUMBER)
RETURN NUMBER
AS
V_PERIOD_OF_SERVICE_YEARS NUMBER;
BEGIN
SELECT EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM TO_DATE(HIRE_DATE)) INTO V_
PERIOD_OF_SERVICE_YEARS
FROM EMPLOYEES
WHERE EMPLOYEE_ID=P_EMP_ID;
RETURN V_PERIOD_OF_SERVICE_YEARS;
END;
/
SELECT EMPLOYEE_ID,FIRST_NAME, EMP_PERIOD_OF_SERVICE_YEAR(EMPLOYEE_ID) AS PERIOD_OF_
SERVICE_YEAR FROM EMPLOYEES;

```

EMPLOYEE_ID	FIRST_NAME	PERIOD_OF_SERVICE_YEAR
174	Ellen	13
166	Sundar	9
130	Mozhe	12
105	David	12
204	Hermann	15
116	Shelli	12
167	Amit	9
172	Elizabeth	10

Create a Package using the CREATE OR REPLACE PACKAGE statement.

```

CREATE OR REPLACE PACKAGE PCK_CHINOOK_REPORTS
AS
PROCEDURE GET_ARTIST_BY_ALBUM(P_ARTIST_ID ALBUM.TITLE%TYPE);
PROCEDURE CUST_INVOICE_BY_YEAR_ANALYZE;
END;

```

Create a new Package using the CREATE OR REPLACE PACKAGE BODY statement.

```

CREATE OR REPLACE PACKAGE BODY PCK_CHINOOK_REPORTS
AS
PROCEDURE GET_ARTIST_BY_ALBUM(P_ARTIST_ID ALBUM.TITLE%TYPE)
IS
V_ARTIST_NAME ARTIST.NAME%TYPE;
BEGIN
SELECT ART.NAME INTO V_ARTIST_NAME
FROM ALBUM ALB JOIN ARTIST ART USING(ARTISTID)
WHERE ALB.TITLE=P_ARTIST_ID;
DBMS_OUTPUT.PUT_LINE('ArtistName: '||V_ARTIST_NAME);
END;

PROCEDURE CUST_INVOICE_BY_YEAR_ANALYZE
AS
V_CUST_GENRES VARCHAR2(200);
BEGIN
FOR V IN(SELECT CUSTOMERID, CUSTNAME, LOW_YEAR, HIGH_YEAR, CUST_AVG FROM TMP_CUST_
INVOICE_ANALYZE)
LOOP
IF SUBSTR(V.LOW_YEAR, -4) > SUBSTR(V.HIGH_YEAR, -4) THEN
SELECT LISTAGG(GENRE, ',') WITHIN GROUP (ORDER BY GENRE) INTO V_CUST_GENRES FROM
(SELECT DISTINCT
FUNC_GENRE_BY_ID(TRC.GENREID) AS GENRE
FROM TMP_CUST_INVOICE_ANALYZE TMPTBL JOIN INVOICE INV USING(CUSTOMERID)
JOIN INVOICELINE INVLIN
ON INV.INVOICEID = INVLIN.INVOICEID
JOIN TRACK TRC
ON TRC.TRACKID = INVLIN.TRACKID
WHERE CUSTOMERID=V.CUSTOMERID);
DBMS_OUTPUT.PUT_LINE('Customer: '||UPPER(V.CUSTNAME)||' - Offer a Discount According
To Preferred Genres: '||UPPER(V_CUST_GENRES));
END IF;
END LOOP;
END;
END;

EXEC PCK_CHINOOK_REPORTS.GET_ARTIST_BY_ALBUM();
EXEC PCK_CHINOOK_REPORTS.CUST_INVOICE_BY_YEAR_ANALYZE;

```

The above examples demonstrate basic Oracle PL/SQL procedure and function capabilities. Oracle PL/SQL provides a large number of features and capabilities that are not within the scope of this document.

For more details, see

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-FUNCTION.html#GUID-156AEDAC-ADD0-4E46-AA56-6D1F7CA63306>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-PROCEDURE.htm-#GUID-771879D8-BBFD-4D87-8A6C-290102142DA3>

MySQL Usage

Aurora MySQL Stored Procedures provide similar functionality to Oracle stored procedures. As with Oracle, Aurora MySQL supports security execution context. It also supports input, output, and bi-directional parameters.

Stored procedures are typically used for:

- **Code Reuse:** Stored procedures provide a convenient code encapsulation and reuse mechanism for multiple applications, potentially written in various languages, requiring the same database operations.
- **Security Management:** By allowing access to base tables only through stored procedures, administrators can manage auditing and access permissions. This approach minimizes dependencies between application code and database code. Administrators can use stored procedures to process business rules and to perform auditing and logging.
- **Performance improvements:** Full SQL query text does not need to be transferred from the client to the database.

Note: Aurora MySQL stored procedures, triggers, and user defined functions are collectively referred to as *Stored Routines*. When binary logging is enabled, MySQL SUPER privilege is required to run stored routines. However, you can run stored routines with binary logging enabled without SUPER privilege by setting the `log_bin_trust_function_creators` parameter to `true` for the DB parameter group for your MySQL instance.

Aurora MySQL permits stored routines to contain control flow, DML, DDL, and transaction management statements including START TRANSACTION, COMMIT, and ROLLBACK.

Syntax

```
CREATE [DEFINER = { user | CURRENT_USER }] PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...]
routine_body

proc_parameter: [ IN | OUT | INOUT ] param_name type

characteristic: COMMENT 'string' | LANGUAGE SQL | [NOT] DETERMINISTIC | { CONTAINS SQL
| NO SQL | READS SQL DATA | MODIFIES SQL DATA } | SQL SECURITY { DEFINER | INVOKER }
```

Examples

The following example demonstrates using a LOOP Cursor with a source table to replace table valued parameters:

Create an OrderItems table.

```
CREATE TABLE OrderItems(OrderID INT NOT NULL, Item VARCHAR(20) NOT NULL, Quantity
SMALLINT NOT NULL, PRIMARY KEY(OrderID, Item));
```

Create and populate SourceTable as a temporary data store for incoming rows.

```
CREATE TABLE SourceTable (OrderID INT, Item VARCHAR(20), Quantity SMALLINT, PRIMARY
KEY (OrderID, Item));
```

```
INSERT INTO SourceTable (OrderID, Item, Quantity) VALUES (1, 'M8 Bolt', 100), (2, 'M8 Nut', 100), (3, 'M8 Washer', 200);
```

Create a procedure to loop through all rows in SourceTable and insert them into the OrderItems table.

```
CREATE PROCEDURE LoopItems ()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE var_OrderID INT;
  DECLARE var_Item VARCHAR(20);
  DECLARE var_Quantity SMALLINT;
  DECLARE ItemCursor CURSOR FOR SELECT OrderID, Item, Quantity FROM SourceTable;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN ItemCursor;
  CursorStart: LOOP
  FETCH NEXT FROM ItemCursor INTO var_OrderID, var_Item, var_Quantity;
  IF Done THEN LEAVE CursorStart;
  END IF;
  INSERT INTO OrderItems (OrderID, Item, Quantity) VALUES (var_OrderID, var_Item,
var_Quantity);
  END LOOP;
  CLOSE ItemCursor;
END;
```

Call the stored procedure.

```
CALL LoopItems ();
```

Select all rows from the OrderItems table.

```
SELECT * FROM OrderItems;
```

OrderID	Item	Quantity
1	M8 Bolt	100
2	M8 Nut	100
3	M8 Washer	200

Summary

The following table summarizes the differences between Aurora MySQL Stored Procedures and Oracle Stored Procedures.



	Oracle	Aurora MySQL	Workaround
General CREATE Syntax differences	CREATE PROCEDURE <Procedure Name> Parameter1 <Type>, ...n AS <Body>	CREATE PROCEDURE <Procedure Name> (Parameter1 <Type>, ...n) <Body>	Rewrite stored procedure creation scripts to use PROCEDURE instead of PROC. Rewrite stored procedure creation scripts to omit the AS keyword.
Security Context	{ AUTHID } { CURRENT_USER	DEFINER = 'user' CURRENT_USER	For Stored procedures that use an explicit user name, rewrite the code

	Oracle	Aurora MySQL	Workaround
	DEFINER}	in conjunction with SQL SECURITY { DEFINER INVOKER }	from EXECUTE AS 'user' to DEFINER = 'user' and SQL SECURITY DEFINER. For Stored Procedures that use the CALLER option, rewrite the code to include SQL SECURITY INVOKER. For Stored procedures that use the SELF option, rewrite the code to DEFINER = CURRENT_USER and SQL SECURITY DEFINER.
Parameter direction	IN and OUT, by default OUT can be used as IN as well.	IN, OUT, and INOUT	

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/faqs-stored-procs.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-procedure.html>

Oracle Regular Expressions vs. MySQL Regular Expressions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Syntax and options differences

Oracle Usage

A Regular Expression is a set of characters that define a search pattern. The most basic example is *, which matches any character. Most Relational Database Management Systems use the same characters for regular expressions, but some use characters differently and provide additional expressions.

Oracle SQL implementation is based on the following standards:

- IEEE Portable Operating System Interface (POSIX) standard draft 1003.2/D11.2
- Unicode Regular Expression Guidelines of the Unicode Consortium

Oracle SQL extends the standards as follows:

- Provides matching capabilities for multilingual data.
- Supports some commonly used PERL regular expression operators not included in the POSIX standard (for example, character class shortcuts and the non-greedy modifier [?]).

Summary of Oracle SQL Pattern-Matching:

- **REGEXP_LIKE**: Can be used in WHERE clauses to find rows matching a certain pattern.
- **REGEXP_COUNT**: Returns the number of occurrences of a pattern in a given string.
- **REGEXP_INSTR**: Returns the position of a pattern within a string.
- **REGEXP_REPLACE**: Replaces a pattern within a string and returns the new string.
- **REGEXP_SUBSTR**: Similar to REGEXP_INSTR, but returns the matching substring itself instead of its position.

Summary of Oracle SQL Pattern-Matching options:

- **i** – Case-insensitive matching
- **c** – Case-sensitive matching.
- **n** – Allows the Dot operator (.) to act like a newline character
- **m** – Allows the string to contain multiple lines
- **x** – Ignores white-space characters in the search pattern

Examples

Find employees with a first name of Steven or Stephen.

```
SELECT * FROM EMPLOYEES
WHERE REGEXP_LIKE((first_name, '^Ste(v|ph)en$')
```

Find employees with a first name that includes 'g' (but not G) twice starting at character position 3.

```
SELECT * FROM EMPLOYEES where
REGEXP_COUNT('George Washington', 'g', 3, 'c') = 2;
```

Find employees with a valid email address.

```
SELECT * FROM EMPLOYEES where
REGEXP_INSTR(email, '\w+@\w+(\.\w+)+') >0;
```

Get the country with a space after each character for each employee.

```
SELECT REGEXP_REPLACE(country_name, '(.)', '\1 ') FROM EMPLOYEES;
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Oracle-Regular-Expression-Support.html#GUID-969230D6-FC1A-4C75-BF2A-6B1BE909DED6>

MySQL Usage

Like Oracle, Aurora MySQL Regular Expressions to make complex searches easier.

MySQL and Oracle use Henry Spencer's implementation of regular expressions, which implements the POSIX 1003.2 standard. MySQL uses the extended version to support regular expression pattern-matching operations in SQL statements.

RDS MySQL Only (not Aurora MySQL): In MySQL 8.0, support for Regular Expressions will be more like Oracle. For more details, see https://dev.mysql.com/doc/refman/5.7/en/regexp.html#function_regexp-replace

Regular Expression Operators

- **NOT REGEXP or NOT RLIKE:** Returns 1 if the string expr does not match the regular expression specified by the pattern pat. Otherwise, it returns 0. If either expr or pat is NULL, the return value is NULL.
- **REGEXP or RLIKE:** Returns 1 if the string expr matches the regular expression specified by the pattern pat. Otherwise, it returns 0. If either expr or pat is NULL, the return value is NULL.

Notes

- RLIKE is a synonym for REGEXP. For compatibility with Oracle, this section refers only to REGEXP.
- MySQL uses the C escape syntax in strings. You must double any '\' used in your REGEXP arguments.

Examples

Find employees with a first name of Steven or Stephen.

```
SELECT * FROM EMPLOYEES WHERE first_name REGEXP ('^Ste(v|ph)en$');
```

Find employees with a valid email address.

```
SELECT * FROM EMPLOYEES where  
email NOT REGEXP '^[A-Z0-9._%~]+@[A-Z0-9.-]+\.[A-Z]{2,4}$';
```



Summary

search or usage	Oracle	MySQL
Find employees with the first name of Steven or Stephen	SELECT * FROM EMPLOYEES WHERE REGEXP_LIKE((first_name, '^Ste(v ph)en\$')	SELECT * FROM EMPLOYEES WHERE first_name REGEXP ('^Ste(v ph)en\$');
Find employees with the first name that includes 'g' (but not G) twice , starting at character position 3	SELECT * FROM EMPLOYEES WHERE REGEXP_COUNT('George Wash- ington', 'g', 3, 'c') = 2;	select * FROM EMPLOYEES WHERE LENGTH(SUBSTRING(FULL_ NAME,3)) - LENGTH(REPLACE (SUBSTRING(FULL_NAME,3), 'g', '')) = 2;
Find employees with a valid email address	SELECT * FROM EMPLOYEES where REGEXP_INSTR(email, '\w+@\w+ (\.\w+)+') >0;	SELECT * FROM EMPLOYEES where email NOT REGEXP '^[A-Z0- 9._%~]+@[A-Z0-9.-]+\.[A-Z]{2,4}\$';
Get each employee's country with space after each character	SELECT REGEXP_REPLACE (country_name, '(.)', '\1 ') FROM EMPLOYEES;	Must be performed with a user defined function

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/regexp.html>
- <https://dev.mysql.com/doc/refman/5.7/en/pattern-matching.html>

Oracle TIMEZONE Data Type and Functions vs. MySQL CONVERT_TZ Function

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Date and time	There is no equivalent option for Oracle's CREATE TABLE...TIMESTAMP WITH TIME ZONE in MySQL but CONVERT_TZ can be used to achieve the same results

Oracle Usage

Oracle uses data types and functions to integrate with time zones. See [A Time Zones](#) for a list of all time zones.

The following data types are variants of TIMESTAMP:

- **TIMESTAMP WITH LOCAL TIME ZONE:** Data stored in the database is normalized to the database time zone, and the time zone offset is not stored as part of the column data. When users retrieve the data, Oracle returns it in the user's local session time zone.
- **TIMESTAMP WITH TIME ZONE:** Includes a time zone offset or time zone region name in its value.

Best practices:

- Use the TIMESTAMP WITH TIME ZONE data type when the application is used across time zones.
- The TIMESTAMP WITH TIME ZONE data type requires 13 bytes of storage; two more bytes of storage than TIMESTAMP WITH LOCAL TIME ZONE data types.

Note: The retrieved time zone offset is the difference (in hours and minutes) between local time and UTC.

Time Zone Functions

Function	Description
NEW_TIME	Converts date and time from one TZ to another.
FROM_TZ	Converts a TZ to a TIMESTAMP WITH TIME ZONE value.
CURRENT_TIMESTAMP	Returns the current date and time in the session time zone.
DBTIMEZONE	Returns the current date and time in the database time zone.
SYS_EXTRACT_UTC	Returns the UTC date and time.
TO_TIMESTAMP_TZ	Converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 to TIMESTAMP WITH TIME ZONE.

Examples

Create a table using `TIMESTAMP WITH LOCAL TIME ZONE`. Note that the last inserted row is displayed as a local session timestamp. It is the only row inserted using a specific time zone that is not LOCAL.

```
CREATE TABLE tz_local
(id NUMBER, tz_col TIMESTAMP WITH LOCAL TIME ZONE);

INSERT INTO tz_local VALUES(1, '01-JAN-2018 2:00:00');
INSERT INTO tz_local VALUES(2, TIMESTAMP '2018-01-01 2:00:00');
INSERT INTO tz_local VALUES(3, TIMESTAMP '2018-01-01 2:00:00 -08:00');

COMMIT;

SELECT * FROM tz_local;
```

ID	TZ_COL
1	2018-01-01 02:00:00
2	2018-01-01 02:00:00
3	2018-01-01 05:00:00

Create a table using `TIMESTAMP WITH TIME ZONE`. Note that the last inserted row is displayed as a local session timestamp. It is the only row that inserted using a specific time zone.

```
ALTER SESSION SET TIME_ZONE='-4:00';

CREATE TABLE tz_tbl (id NUMBER, tz_col TIMESTAMP WITH TIME ZONE);

INSERT INTO tz_tbl VALUES(1, '01-JAN-2018 2:00:00 AM -5:00');
INSERT INTO tz_tbl VALUES(2, TIMESTAMP '2018-01-01 3:00:00');
INSERT INTO tz_tbl VALUES(3, TIMESTAMP '2018-01-01 2:00:00 -8:00');

COMMIT;

SELECT * FROM tz_tbl;
```

ID	TZ_COL
1	01-JAN-03 02:00:00.000000 AM -07:00
2	01-JAN-03 02:00:00.000000 AM -07:00
3	01-JAN-03 02:00:00.000000 AM -08:00

MySQL Usage

MySQL uses timezone data type and functions similar to Oracle. Unlike Oracle, MySQL does not have many time zone options. Most functionality can be achieved when querying and not when running DDLs (like Oracle's `CREATE TABLE` command).

When the server starts, it places the host timezone in the `system_time_zone` system variable. This variable can be modified by setting the timezone operating system environment variable.

There is no equivalent option for Oracle's `CREATE TABLE...TIMESTAMP WITH TIME ZONE`.

Comparison of Time Zone Functions

Oracle Function	MySQL Function
NEW_TIME	CONVERT_TZ can be used, but you have to specify the source TZ.
FROM_TZ	CONVERT_TZ
DBTIMEZONE	CONVERT_TZ(CURRENT_TIME(),@@global.time_zone,@@global.time_zone)
SYS_EXTRACT_UTC	CONVERT_TZ(CURRENT_TIME(),@@global.time_zone,'+00:00')
TO_TIMESTAMP_TZ	CONVERT_TZ(STR_TO_DATE('17-09-2010 23:15','%d-%m-%Y %H:%i'),@@global.time_zone,'+03:00')

Examples

Query the global and session level time zone.



```
SELECT @@global.time_zone, @@session.time_zone;

@@global.time_zone |@@session.time_zone |
-----|-----|
SYSTEM            |Europe/Moscow       |
```

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/time-zone-support.html>
- <https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html>

Oracle User-Defined Functions vs. MySQL User-Defined Functions

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Functions	Syntax and options differences

Oracle Usage

You can create an Oracle User-Defined Function (UDF) using PL/SQL, Java, or C. UDFs are useful for providing functionality not available in SQL or SQL built-in functions. They can appear in SQL statements wherever built-in SQL functions can appear.

- UDFs can be used to return a single value from a SELECT statement (scalar function).
- UDFs can be used while performing DML operations.
- UDFs can be used in WHERE, GROUP BY, ORDER BY, HAVING, CONNECT BY, and START WITH clauses.

Examples

Create a simple Oracle UDF with arguments for employee HIRE_DATE and SALARY as INPUT parameters and calculate the overall salary over the employee's years of service for the company.

```
CREATE OR REPLACE FUNCTION TOTAL_EMP_SAL_BY_YEARS
(p_hire_date DATE, p_current_sal NUMBER)
RETURN NUMBER
AS
v_years_of_service NUMBER;
v_total_sal_by_years NUMBER;
BEGIN
SELECT EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM to_date(p_hire_date))
INTO v_years_of_service FROM dual;
v_total_sal_by_years:=p_current_sal*v_years_of_service;
RETURN v_total_sal_by_years;
END;
/
-- Verifying
SELECT EMPLOYEE_ID, FIRST_NAME, TOTAL_EMP_SAL_BY_YEARS(HIRE_DATE, SALARY)AS TOTAL_
SALARY
FROM EMPLOYEES;
```

EMPLOYEE_ID	FIRST_NAME	TOTAL_SALARY
100	Steven	364000
101	Neena	204000
102	Lex	272000
103	Alexander	99000
104	Bruce	60000

```
105 David 57600
```

```
...
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-FUNCTION.htm-#GUID-156AEDAC-ADD0-4E46-AA56-6D1F7CA63306>

MySQL Usage

Aurora MySQL supports User-defined scalar functions only. There is no support for table-valued functions.

Aurora MySQL does not permit stored functions to contain explicit SQL transaction statements such as COMMIT and ROLLBACK.

In Aurora MySQL, you can explicitly specify several options with the CREATE FUNCTION statement. These characteristics are saved with the function definition and are viewable with the SHOW CREATE FUNCTION statement.

- The DETERMINISTIC option must be explicitly stated. Otherwise, the engine assumes it is not deterministic.

Note: MySQL does not check the validity of the deterministic property declaration. If you wrongly specify a function as DETERMINISTIC when it is not, unexpected results and errors may occur.

- CONTAINS SQL indicates the function code does not contain statements that read or modify data.
- READS SQL DATA indicates the function code contains statements that read data (for example, SELECT) but not statements that modify data (for example, INSERT, DELETE, or UPDATE).
- MODIFIES SQL DATA indicates the function code contains statements that may modify data.

Note: The above options are advisory only. The server does not constrain the function code based on the declaration. This feature is useful for code management.

Syntax

```
CREATE FUNCTION <Function Name> ([<Function Parameter>[,...]])
RETURNS <Returned Data Type> [characteristic ...]
<Function Code Body>

characteristic:
COMMENT '<Comment>' | LANGUAGE SQL | [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

Migration Considerations

For scalar functions, migration should be straightforward with respect to the function syntax. Rules in Aurora MySQL regarding functions are much more lenient than Oracle.

A function in Aurora MySQL may modify data and schema. Function determinism must be explicitly stated, unlike Oracle that infers it from the code. Additional properties can be stated for a function, but most are advisory only and have no functional impact.

The AS keyword, which is mandatory in Oracle before the function's code body, is not valid Aurora MySQL syntax and must be removed.

Examples

Create a scalar function to change the first character of a string to upper case.

```
CREATE FUNCTION UpperCaseFirstChar (String VARCHAR(20))
RETURNS VARCHAR(20)
BEGIN
RETURN CONCAT(UPPER(LEFT(String, 1)) , LOWER(SUBSTRING(String, 2, 19)));
END

SELECT UpperCaseFirstChar ('mIxEdCasE');

-----
Mixedcase
```

Summary



The following table identifies similarities, differences, and key migration considerations.

Oracle	Aurora MySQL	Comment
Scalar UDF	Scalar UDF	Use CREATE FUNCTION with similar syntax, remove the AS keyword.
Inline table valued UDF	N/A	Use views and replace parameters with WHERE filter predicates.
Multistatement table valued UDF	N/A	Use stored procedures to populate tables and read from the table directly.
UDF determinism implicit	Explicit declaration	Use the DETERMINISTIC characteristic explicitly to denote a deterministic function, which enables engine optimizations.
UDF boundaries local only	Can change data and schema	UDF rules are more lenient, avoid unexpected changes from function invocation.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-procedure.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-function-udf.html>

Oracle UTL_FILE vs. MySQL Integration with Amazon S3

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL does not support UTL_FILE but Aurora MySQL has a built-in integration with Amazon S3

Oracle Usage

The Oracle UTL_FILE PL/SQL package enables you to access files stored outside of the database such as files stored on the operating system, the database server, or a connected storage volume. The UTL_FILE.FOPEN, UTL_FILE.GET_LINE, and UTL_FILE.PUT_LINE are procedures within the UTL_FILE package used to open, read, and write files.

Examples

Run an anonymous PL/SQL block that reads a single line from file1 and writes it to file2.

- Use UTL_FILE.FILE_TYPE to create a handle for the file.
- Use UTL_FILE.FOPEN to open streamable access to the file and specify:
 - The logical Oracle directory object pointing to the O/S folder where the file resides
 - The file name
 - The file access mode: 'A'=append mode, 'W'=write mode
- Use UTL_FILE.GET_LINE to read a line from the input file into a variable.
- Use UTL_FILE.PUT_LINE to write a single line to the output file.

```

DECLARE
strString1 VARCHAR2(32767);
fileFile1 UTL_FILE.FILE_TYPE;
BEGIN
fileFile1 := UTL_FILE.FOPEN('FILES_DIR','File1.tmp','R');
UTL_FILE.GET_LINE(fileFile1,strString1);
UTL_FILE.FCLOSE(fileFile1);
fileFile1 := UTL_FILE.FOPEN('FILES_DIR','File2.tmp','A');
utl_file.PUT_LINE(fileFile1,strString1);
utl_file.fclose(fileFile1);
END;
/

```

For additional details, see: For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/UTL_FILE.html#GUID-EBC42A36-EB72-4AA1-B75F-8CF4BC6E29B4

MySQL Usage

Aurora MySQL provides similar functionality to Oracle's UTL_FILE with Amazon S3 integration.

There two important integration aspects between Aurora MySQL and Amazon S3:

- Saving data to an S3 file.
- Loading data from an S3 file.

Note: Aurora MySQL must have permissions to the S3 bucket. See the links at the bottom of this section for more information.

Saving Data to Amazon S3

You can use the SELECT INTO OUTFILE S3 statement to query data from an Amazon Aurora MySQL DB cluster and save it directly to text files stored in an Amazon S3 bucket. You can use this approach to avoid transferring data first to the client and then copying the data from the client to Amazon S3.

Note: The default file size threshold is 6 GB. If the data selected by the statement is less than the file size threshold, a single file is created. Otherwise, multiple files are created.

If the SELECT statement failed, files already uploaded to Amazon S3 remain in the specified Amazon S3 bucket. You can use another statement to upload the remaining data instead of starting over.

If the amount of data to be selected is large (more than 25 GB), it is recommended to use multiple SELECT INTO OUTFILE S3 statements to save the data to Amazon S3.

Metadata, such as table schema or file metadata, is not uploaded by Aurora MySQL to Amazon S3.

Examples

The following statement selects all data in the employees table and saves it to an Amazon S3 bucket in a different region from the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. The statement returns an error if files that match the sample_employee_data file prefix already exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-  
pdx/sample_employee_data'  
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
```

The following statement selects all data in the employees table and saves the data to an Amazon S3 bucket in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. It also creates a manifest file. The statement returns an error if files that match the sample_employee_data file prefix already exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/sample_  
employee_data'  
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' MANIFEST ON;
```

The following statement selects all data in the employees table and saves the data to an Amazon S3 bucket in a different region from the Aurora database cluster. The statement creates data files in which each field is ter-

minated by a comma (,) character and each row is terminated by a newline (\n) character. The statement overwrites any existing files that match the sample_employee_data file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-
pdx/sample_employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' OVERWRITE ON;
```

The following statement selects all data in the employees table and saves the data to an Amazon S3 bucket in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. It also creates a manifest file. The statement overwrites any existing files that match the sample_employee_data file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/sample_
employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
MANIFEST ON OVERWRITE ON;
```

For more information, see [Saving Data from an Amazon Aurora MySQL DB Cluster into Text Files in an Amazon S3 Bucket](#)

LOAD XML FROM AMAZON S3

Use the LOAD DATA FROM S3 or LOAD XML FROM S3 statement to load data from files stored in an Amazon S3 bucket.

The LOAD DATA FROM S3 statement can load data from any text file format supported by the MySQL LOAD DATA INFILE statement such as comma-delimited text data. Compressed files are not supported.

Examples

The following example runs the LOAD DATA FROM S3 statement with the manifest file named customer.manifest. After the statement completes, an entry for each successfully loaded file is written to the aurora_s3_load_history table.

```
LOAD DATA FROM S3 MANIFEST
's3-us-west-2://aurora-bucket/customer.manifest' INTO TABLE CUSTOMER FIELDS TERMINATED
BY ',' LINES TERMINATED BY '\n'
(ID, FIRSTNAME, LASTNAME, EMAIL);
```

Every successful LOAD DATA FROM S3 statement updates the aurora_s3_load_history table in the mysql schema with an entry for each file that was loaded.

After you run the LOAD DATA FROM S3 statement, you can verify which files were loaded by querying the aurora_s3_load_history table. To see the files that were loaded from one execution of the statement, use the WHERE clause to filter the records on the Amazon S3 URI for the manifest file used in the statement. If you have used the same manifest file before, filter the results using the timestamp field.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

The following table describes the fields in the aurora_s3_load_history table.

Field	Description
load_prefix	The URI specified in the load statement. This URI can map to any of the following: <ul style="list-style-type: none"> • A single data file for a LOAD DATA FROM S3 FILE statement. • An Amazon S3 prefix that maps to multiple data files for a LOAD DATA FROM S3 PREFIX statement. • A single manifest file containing the names of files to be loaded for a LOAD DATA FROM S3 MANIFEST statement.
file_name	The name of a file that was loaded into Aurora from Amazon S3 using the URI identified in the load_prefix field.
version_number	The version number of the file identified by the file_name field that was loaded (if the Amazon S3 bucket has a version number).
bytes_loaded	The size in bytes of the file loaded.
load_timestamp	The timestamp when the LOAD DATA FROM S3 statement completed.

Examples

The following statement loads data from an Amazon S3 bucket in the same region as the Aurora DB cluster. It reads the comma-delimited data in the file customerdata.txt residing in the dbbucket Amazon S3 bucket and then loads the data into the table store-schema.customer-table.

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv' INTO TABLE
store-schema.customer-table FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n' (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

The following statement loads data from an Amazon S3 bucket in a different region from the Aurora DB cluster. The statement reads the comma-delimited data from all files that match the employee-data object prefix in the my-data Amazon S3 bucket in the us-west-2 region and then loads the data into the employees table.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data' INTO TABLE employees
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (ID, FIRSTNAME, LASTNAME, EMAIL,
SALARY);
```

The following statement loads data from the files specified in a JSON manifest file named q1_sales.json into the sales table.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json' INTO TABLE
sales FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

You can use the LOAD XML FROM S3 statement to load data from XML files stored on an Amazon S3 bucket in one of three different XML formats as described below.

Column names as attributes of a <row> element. The attribute value identifies the contents of the table field.

```
<row column1="value1" column2="value2" .../>
```

Column names as child elements of a <row> element. The value of the child element identifies the contents of the table field.

```
<row>
<column1>value1</column1>
<column2>value2</column2>
</row>
```

Column names in the name attribute of <field> elements in a <row> element. The value of the <field> element identifies the contents of the table field.

```
<row>
<field name='column1'>value1</field>
<field name='column2'>value2</field>
</row>
```

The following statement loads the first column from the input file into the first column of table1 and sets the value of the table_column2 column in table1 to the input value of the second column divided by 100.

```
LOAD XML FROM S3 's3://mybucket/data.xml' INTO TABLE table1 (column1, @var1) SET
table_column2 = @var1/100;
```



The following statement sets the first two columns of table1 to the values in the first two columns from the input file and then sets the value of the column3 in table1 to the current time stamp.

```
LOAD XML FROM S3 's3://mybucket/data.xml' INTO TABLE table1 (column1, column2) SET
column3 = CURRENT_TIMESTAMP;
```

You can use subqueries in the right side of SET assignments. For a subquery that returns a value to be assigned to a column, you can use only a scalar subquery. Also, you cannot use a subquery to select from the table that is being loaded.

For more details, see [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket](#)

Oracle UTL_MAIL & UTL_SMTP(11g & 12c) vs. MySQL Amazon RDS and Amazon Simple Notification Service (SNS)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Use AWS Lambda Integration

Oracle Usage UTL_MAIL

The Oracle UTL_MAIL package provides functionality for sending email messages. Unlike UTL_SMTP, which is more complex and provided in earlier versions of Oracle, UTL_MAIL supports attachments. For most cases, UTL_Mail is a better choice.

Examples

Install the required mail packages.

```
@{ORACLE_HOME}/rdbms/admin/utlmail.sql
@{ORACLE_HOME}/rdbms/admin/prvtmail.plb
```

Set the smtp_out_server parameter.

```
ALTER SYSTEM SET smtp_out_server = 'smtp.domain.com' SCOPE=BOTH;
```

Send an email message.

```
exec utl_mail.send('Sender@mailserver.com', 'recipient@mailserver.com', NULL, NULL,
'This is the subject', 'This is the message body', NULL, 3, NULL);
```

For more details, see [UTL_MAIL](#)

Usage UTL_SMTP

The Oracle UTL_SMTP package provides functionality for sending email messages and is useful for sending alerts about database events. Unlike UTL_MAIL, UTL SMTP is more complex and does not support attachments. For most cases, UTL_Mail is a better choice.

Examples

The following example demonstrates using UTL_SMTP procedures to send email messages:

Install the required scripts.

```
In oracle 12c:
@{ORACLE_HOME}/rdbms/admin/utlsmtp.sql
```

```
In oracle 11g:
@{ORACLE_HOME}/javavm/install/initjvm.sql
@{ORACLE_HOME}/rdbms/admin/initplsjsql
```

Create and send an email message:

- UTL_SMTP.OPEN_CONNECTION opens a connection to the smtp server.
- UTL_SMTP.HELO initiates a handshake with the smtp server.
- UTL_SMTP.MAIL Initiates a mail transaction that obtains the senders details.
- UTL_SMTP.RCPT adds a recipient to the mail transaction.
- UTL_SMTP.DATA adds the message content.
- UTL_SMTP.QUIT terminates the SMTP transaction.

```
DECLARE
smtpconn utl_smtp.connection;
BEGIN
smtpconn := UTL_SMTP.OPEN_CONNECTION('smtp.mailserver.com', 25);
UTL_SMTP.HELO(smtpconn, 'smtp.mailserver.com');
UTL_SMTP.MAIL(smtpconn, 'sender@mailserver.com');
UTL_SMTP.RCPT(smtpconn, 'recipient@mailserver.com');
UTL_SMTP.DATA(smtpconn, 'Message body');
UTL_SMTP.QUIT(smtpconn);
END;
/
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-resources-with-oracle-database-resource-manager.html#GUID-2BEF5482-CF97-4A85-BD90-9195E41E74EF>

MySQL Usage

Aurora MySQL does not support direct configuration of engine alerts. Use the Event Notifications Infrastructure to collect history logs or receive event notifications in near real-time.

The Amazon Relational Database Service (Amazon RDS) uses the Amazon Simple Notification Service (SNS) to provide notifications for events. Amazon SNS can send notifications in any form supported by the region including email, text messages, or calls to HTTP endpoints for response automation.

Events are grouped into categories. You can only subscribe to event categories, not individual events. SNS sends notifications when any event in a category occurs.

You can subscribe to alerts for database instances, database clusters, database snapshots, database cluster snapshots, database security groups, and database parameter groups. For example, a subscription to the *Backup* category for a specific database instance sends notifications when backup-related events occur on that instance. A subscription to the *Configuration Change* category for a database security group sends notifications when the security group changes.

Note: For Amazon Aurora, some events occur at the cluster rather than instance level. You will not receive those events if you subscribe to an Aurora DB instance.

SNS sends event notifications to the address specified when the subscription was created. Typically, administrators create several subscriptions. For example, one subscription to receive logging events and another to receive only critical events for a production environment requiring immediate responses.

You can disable notifications without deleting a subscription by setting the *Enabled* radio button to *No* in the Amazon RDS console. Alternatively, use the AWS Command Line Interface (CLI) or Amazon RDS API to change the *Enabled* setting.

Subscriptions are identified by the Amazon Resource Name (ARN) of an Amazon SNS topic. The Amazon RDS console creates ARNs when subscriptions are created. When using the CLI or API, you must create the ARN using the Amazon SNS console or the Amazon SNS API.

Examples

The following walkthogh demonstrates how to create an Event Notification Subscription:

1. Sign into an Amazon AWS account, open the AWS Console, and navigate to the Amazon RDS page.
2. In the left pane, select Events.

The screenshot displays the Amazon RDS console interface. On the left, a navigation sidebar lists various RDS-related features, with 'Events' selected. The main content area is divided into two sections. The top section, titled 'Amazon Aurora', includes an information icon, a description of Aurora as a MySQL- and PostgreSQL-compatible database, and a prominent orange 'Create database' button. Below this, there is a link to 'Restore Aurora DB cluster from S3'. The bottom section, titled 'Resources', features a 'Refresh' button and text indicating resource usage in the EU (Frankfurt) region. It lists 'DB Instances (4/40)' with a link to increase instances, and 'Parameter groups (6)' with sub-counts for 'Default (5)' and 'Custom (1/100)'.

If you have not previously subscribed to events, the screen displays zero events.

RDS > Events

Events (8) 🔄

🔍 < 1 > ⚙️

Source	Type	Time	Message
rds:mysql-aurora-playbook-2021-02-26-02-32	Cluster Snapshots	February 26, 2021, 2:32:37 AM UTC	Automated cluster snapshot created
rds:mysql-aurora-playbook-2021-02-26-02-32	Cluster Snapshots	February 26, 2021, 2:32:33 AM UTC	Creating automated cluster snapshot
rds:pg-playbooks-2021-02-25-21-26	Cluster Snapshots	February 25, 2021, 9:27:19	Automated cluster snapshot created

3. Select **Event subscriptions**.

4. In the right pane, select **Create event subscription**.

5. Enter **Name of the subscription** and select a **Target** of ARN or New email topic. For email subscriptions, enter values for **Topic name** and **With these recipients**.

RDS > Event subscriptions > Create event subscription

Create event subscription

Details

Name
Name of the subscription

Must contain from 1 to 255 characters. Alphanumeric characters and hyphens are allowed.

Target

Send notifications to

ARN

New email topic

ARN
ARN to send notifications to

Choose ARN ▼

Source

Source Type
Source type of resource this subscription will consume events from

Choose source type ▼

Cancel **Create**

6. Select the event source and choose specific event categories. Click the drop-down menu to view the list of available categories.

Q |

- availability
- backup
- configuration change
- creation
- deletion
- failover
- failure
- low storage
- maintenance
- notification
- read replica
- recovery
- restoration

select event categories ▼

7. Choose the event categories to be monitored and select **Create**.

Source

Source type
Source type of resource this subscription will consume event from

Instances

Instances to include
Instances that this subscription will consume events from

All instances
 Select specific instances

Event categories to include
Event categories that this subscription will consume events from

All event categories
 Select specific event categories

Specific event

select event categories

configuration change × low storage ×

Cancel **Create**

8. On the Amazon RDS dashboard, select **Recent events** .

Dashboard

- Databases
- Query Editor
- Performance Insights
- Snapshots
- Automated backups
- Reserved instances
- Proxies

- Subnet groups
- Parameter groups
- Option groups

- Events
- Event subscriptions

Resources

You are using the following Amazon RDS (used/quota)

DB Instances (4/40)
Allocated storage (0.02 TB/100 TB)
[Click here to increase DB instances limit](#)

DB Clusters (2/40)

Reserved instances (0/40)

Snapshots (6)
Manual (0/100)
Automated (6)

Recent events (8)

Event subscriptions (0/20)

For more information, see [Using Amazon RDS Event Notification](#)

Notes

For application email requirements, consider using a dedicated email framework. If the code generating email messages must reside in the database, consider using a queue table. Replace all occurrences of UTL_SMTP and UTL_MAIL with an INSERT into the queue table. Design external applications to connect, read the queue, send an email message, and then update the status periodically. With this approach, messages can be populated with a query result similar to UTL_SMTP and UTL_MAIL with the query option.

The only way to send Email from the database is to use AWS Lambda integration. For more information about AWS Lambda, see [AWS Lambda](#).

For an example of sending an email message from Aurora MySQL using AWS Lambda integration, see [Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster](#)

Tables & Indexes



Case Sensitivity Differences for Oracle and Aurora MySQL

Object name case sensitivity is different for Oracle and Aurora MySQL. Oracle names are not case sensitive. Aurora MySQL names are case sensitive.

In Aurora for MySQL, the case sensitivity is determined by the `lower_case_table_names` parameter's value.

In general there are 3 possible values for this parameter but in order to avoid some issues, Amazon allows only 2 values to be used with this parameter:

- 0 ("names stored as given and comparisons are case-sensitive") is supported for all Amazon RDS for MySQL versions.
- 1 ("names stored in lowercase and comparisons are not case-sensitive") is supported for Amazon RDS for MySQL version 5.6, version 5.7, and version 8.0.19 and higher 8.0 versions.

The `lower_case_table_names` parameter should be set as part of a custom DB parameter group before creating a DB instance. You should avoid changing the `lower_case_table_names` parameter for existing database instances because doing so could cause inconsistencies with point-in-time recovery backups and read replica DB instances.

Read replicas should always use the same `lower_case_table_names` parameter value as the source DB instance.

By default, object names are being stored in lower-case for MySQL. In most cases, you'll want to use AWS DMS transformations to change schema, table, and column names to lower case.

For example, to create a table named EMPLOYEES (upper-case) in MySQL, you should use the following (this option is also valid for Oracle and using double-quotes):

```
CREATE TABLE "EMPLOYEES" (  
    EMP_ID          NUMERIC PRIMARY KEY,  
    EMP_FULL_NAME  VARCHAR(60) NOT NULL,  
    AVG_SALARY     NUMERIC NOT NULL);
```

The command below will create a table named employees (lower-case)



```
CREATE TABLE employees (  
    EMP_ID          NUMERIC PRIMARY KEY,  
    EMP_FULL_NAME  VARCHAR(60) NOT NULL,  
    AVG_SALARY     NUMERIC NOT NULL);
```

MySQL will look for objects names in with the exact case sensitivity as written in the query

You can disable table name case sensitivity in MySQL by setting the parameter `lower_case_table_names` to 1. Column, index, stored routine, event names, and column aliases are not case sensitive on either platform.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/identifier-case-sensitivity.html>

Oracle Common Data Types vs. MySQL Common Data Types

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Data Type	BFILE, ROWID, UROWID are not supported in Aurora MySQL

Oracle Usage

Oracle provides a set of primitive data types for defining table columns and PL/SQL code variables. The assigned data types for table columns or PL/SQL code (such as stored procedures and triggers) define the valid values each column or argument can store.

Oracle Data Types vs. MySQL Data Types

Oracle Data Type Family	Oracle Data Type	Oracle Data Type Characteristic	MySQL Identical Compatibility	MySQL Corresponding Data Type
Character Data Types	CHAR(n)	Maximum size of 2000 bytes	Yes	CHAR(n)
	CHARACTER(n)	Maximum size of 2000 bytes	Yes	CHARACTER(n)
	NCHAR(n)	Maximum size of 2000 bytes	Yes	NCHAR(n)
	VARCHAR(n)	Maximum size of 2000 bytes	Yes	VARCHAR(n)
	NCHAR VARYING(n)	Varying-length UTF-8 string Maximum size of 4000 bytes	Yes	NCHAR VARYING(n)
	VARCHAR2(n) 11g	Maximum size of 4000 bytes Maximum size of 32KB in PL/SQL	No	VARCHAR(n)
	VARCHAR2(n) 12g	Maximum size of 32767 bytes MAX_STRING_SIZE= EXTENDED	No	VARCHAR(n)
	NVARCHAR2(n)	Maximum size of 4000 bytes	No	VARCHAR(n)
	LONG	Maximum size of 2GB	Yes	LONG
	RAW(n)	Maximum size of 2000 bytes	No	VARBINARY(n)
LONG RAW	Maximum size of 2GB	No	LONGTEXT	

Oracle Data Type Family	Oracle Data Type	Oracle Data Type Characteristic	MySQL Identical Compatibility	MySQL Corresponding Data Type
Numeric Data Types	NUMBER	Floating-point number	No	DECIMAL(p,s)
	NUMBER(*)	Floating-point number	No	DOUBLE
	NUMBER(p,s)	Precision can range from 1 to 38 Scale can range from -84 to 127	No	DECIMAL(p,s)
	NUMERIC(p,s)	Precision can range from 1 to 38	Yes	NUMERIC(p,s)
	FLOAT(p)	Floating-point number	Yes	FLOAT(p)
	DEC(p,s)	Fixed-point number	Yes	DEC(p,s)
	DECIMAL(p,s)	Fixed-point number	Yes	DECIMAL(p,s)
	INT	38 digits integer	Yes	INT
	INTEGER	38 digits integer	Yes	INTEGER
	SMALLINT	38 digits integer	Yes	SMALLINT
	REAL	Floating-point number	Yes	REAL
	DOUBLE PRECISION	Floating-point number	Yes	DOUBLE PRECISION
Date & Time Data Types	DATE	DATE data type stores date and time data (year, month, day, hour, minute and second)	Yes	DATE
Date & Time Data Types	TIMESTAMP (p)	Date and time with fraction	Yes	6
	TIMESTAMP (p) WITH TIME ZONE	Date and time with fraction and time zone	No	DATETIME(n)
	INTERVAL YEAR(p) TO MONTH	Date interval	No	VARCHAR(n)
	INTERVAL DAY(p) TO SECOND(s)	Day and time interval	No	VARCHAR(n)
LOB Data Types	BFILE	Pointer to binary file Maximum file size of 4G	No	VARCHAR (255)
	BLOB	Binary large object Maximum file size of 4G	Yes	BLOB
	CLOB	Character large object Maximum file size of 4G	No	LONGTEXT
	NCLOB	Variable-length Unicode string Maximum file size of 4G	No	LONGTEXT
ROWID Data Types	ROWID	Physical row address	No	CHAR(n)
	UROWID(n)	Universal row id Logical row addresses	No	VARCHAR(n)
XML Data Type	XMLTYPE	XML data	No	LONGTEXT

Oracle Data Type Family	Oracle Data Type	Oracle Data Type Characteristic	MySQL Identical Compatibility	MySQL Corresponding Data Type
Logical Data Type	BOOLEAN	Values TRUE / FALSE and NULL Cannot be assign to a database table column	Yes	BOOLEAN
Spatial Types	SDO_GEOMETRY	The geometric description of a spatial object	No	-
	SDO_TOPO_GEOMETRY	Describes a topology geometry	No	-
	SDO_GEORASTER	A raster grid or image object is stored in a single row	No	-
Media Types	ORDDicom	Supports the storage and management of audio data	No	-
	ORDDicom	Supports the storage and management of Digital Imaging and Communications in Medicine (DICOM),	No	-
	ORDDoc	Supports storage and management of any type of media data	No	-
	ORDImage	Supports the storage and management of image data	No	-
	ORDVideo	Supports the storage and management of video data	No	-

Oracle Character Column Semantics

Oracle supports both BYTE and CHAR semantics for column size, which determines the amount of storage allocated for CHAR and VARCHAR columns.

- Fields defined as VARCHAR2(10 BYTE) can use up to 10 bytes for storage. However, based on your database codepage and NLS settings, you may not be able to store 10 characters in that field because the physical size of some non-English characters exceeds one byte.
- Fields defined as VARCHAR2(10 CHAR) can store 10 characters regardless of how many bytes are required to store each non-English character.

```
CREATE TABLE table1 (col1 VARCHAR2(10 CHAR), col2 VARCHAR2(10 BYTE));
```

By default, Oracle uses BYTE semantics. When using a multi-byte character set such as UTF8, you must do one of the following:

- Use the CHAR modifier in the VARCHAR2/CHAR column definition.
- Modify the session or system parameter NLS_LENGTH_SEMANTICS to change the default from BYTE to CHAR.

```
ALTER system SET nls_length_semantics=char scope=both;
ALTER system SET nls_length_semantics=byte scope=both;

ALTER session SET nls_length_semantics=char;
ALTER session SET nls_length_semantics=byte;
```

For more details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Data-Types.html#GUID-A3C0D836-BADB-44E5-A5D4-265BA5968483>

MySQL Usage

MySQL provides multiple data types equivalent to certain Oracle data types. The following table provides the full list of MySQL data types:

MySQL Data Type Family	MySQL Data Type	MySQL Data Type Characteristic
Character Data Types	CHAR(n)	Stores exactly (n) characters.
	VARCHAR(n)	Stores a variable number of characters, up to a maximum of n characters.
	BINARY	Stores exactly (n) bytes.
	VARBINARY	Stores a variable number of characters, up to a maximum of n bytes.
	BLOB	Binary large object that can hold a variable amount of data.
	TEXT	Specific variant of varchar, which does not require you to specify an upper limit on the number of characters.
	ENUM	String object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.
	SET	String object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created.
Numeric Data Types	INTEGER	Max value is 2147483647.
	INT	Max value is 2147483647.
	SMALLINT	Max value is 32767.
	TINYINT	Max value is 127.
	MEDIUMINT	Max value is 8388607.
	BIGINT	Max value is $2^{63}-1$.
	DECIMAL (p,s)	Able to store any value with p digits and s decimals.
	NUMERIC(p,s)	Able to store any value with p digits and s decimals.
	FLOAT (m,d)	Values can be stored with up to M digits in total, of which D digits may be after the decimal point.
	DOUBLE (m,d)	Values can be stored with up to M digits in total, of which D digits may be after the decimal point.
	BIT (m)	Storage of M-bit values. M can range from 1 to 64.
Date & Time Data Types	DATE	Values with a date part but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.
	DATETIME	Values that contain both date and time parts. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
	TIMESTAMP	Values that contain both date and time parts. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.
	TIME	Values may range from '-838:59:59' to '838:59:59'. The hours part may be so large because the TIME type can be used not only

MySQL Data Type Family	MySQL Data Type	MySQL Data Type Characteristic
		to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).
	YEAR	YEAR 1-byte type used to represent year values. It can be declared as YEAR or YEAR(n) and has a display width of n characters.
Logical Data Type	BOOLEAN	Holds a truth value. Will accept values such as TRUE, '1', 1 as true. Uses 1 byte of storage, and can store NULL.
Geometric Data Types	GEOMETRY	The column type to specify when you want to use the data models below.
	POINT	An (x,y) value.
	LINestring	A line (pt1, pt2).
	POLYGON	A sequence of points, effectively a closed path.
Geometric Data Types	MULTIPOINT	Collection of POINTs.
	MULTILINestring	Collection of LINEs.
	MULTIPOLYGON	Collection of POLYGONs.
	GEOMETRYCOLLECTION	Collection of geomery data types.
Others	JSON	Textual JSON data.

Migration of Oracle Data Types to MySQL Data Types

Automatic migration and conversion of Oracle Tables and Data Types can be performed using AWS Schema Conversion Tool (SCT).

Examples

To demonstrate SCT's capability for migrating Oracle tables to their MySQL equivalents, a table containing columns representing the majority of Oracle data types is created and then converted by the tool.

Source Oracle compatible DDL for creating the DATATYPES table:

```
CREATE TABLE "DATATYPES" (
"BFILe"          BFILE,
"BINARY_FLOAT"  BINARY_FLOAT,
"BINARY_DOUBLE" BINARY_DOUBLE,
"BLOB"          BLOB,
"CHAR"          CHAR(10 BYTE),
"CHARACTER"     CHAR(10 BYTE),
"CLOB"          CLOB,
"NCLOB"         NCLOB,
"DATE"          DATE,
"DECIMAL"       NUMBER(3,2),
"DEC"           NUMBER(3,2),
"DOUBLE_PRECISION" FLOAT(126),
"FLOAT"         FLOAT(3),
```

```

"INTEGER"          NUMBER (*, 0),
"INT"              NUMBER (*, 0),
"INTERVAL_YEAR"   INTERVAL YEAR(4) TO MONTH,
"INTERVAL_DAY"    INTERVAL DAY(4) TO SECOND(4),
"LONG"            LONG,
"NCHAR"           NCHAR(10),
"NCHAR_VARYING"   NVARCHAR2(10),
"NUMBER"          NUMBER(9,9),
"NUMBER1"         NUMBER(9,0),
"NUMBER(*)"       NUMBER,
"NUMERIC"         NUMBER(9,9),
"NVARCHAR2"       NVARCHAR2(10),
"RAW"             RAW(10),
"REAL"            FLOAT(63),
"ROW_ID"          ROWID,
"SMALLINT"        NUMBER(*,0),
"TIMESTAMP"       TIMESTAMP(5),
"TIMESTAMP_WITH_TIME_ZONE"
TIMESTAMP(5) WITH TIME ZONE,
"UROWID"          UROWID(10),
"VARCHAR"         VARCHAR2(10 BYTE),
"VARCHAR2"        VARCHAR2(10 BYTE),
"XMLTYPE"         XMLTYPE);

```

Target MySQL compatible DDL for creating the DATATYPES table migrated from Oracle with SCT.

```

CREATE TABLE IF NOT EXISTS datatypes (
bfile          VARCHAR(1000) DEFAULT NULL,
BINARY_FLOAT   FLOAT(12) DEFAULT NULL,
BINARY_DOUBLE  DOUBLE DEFAULT NULL,
`BLOB`         LONGBLOB DEFAULT NULL,
`CHAR`         CHAR(10) DEFAULT NULL,
`CHARACTER`    CHAR(10) DEFAULT NULL,
CLOB           LONGTEXT DEFAULT NULL,
NCLOB          LONGTEXT DEFAULT NULL,
`DATE`        DATETIME DEFAULT NULL,
`DECIMAL`     DECIMAL(3,2) DEFAULT NULL,
`DEC`         DECIMAL(3,2) DEFAULT NULL,
DOUBLE_PRECISION
DOUBLE         DOUBLE DEFAULT NULL,
`FLOAT`       DOUBLE DEFAULT NULL,
`INTEGER`     DECIMAL(38,0) DEFAULT NULL,
`INT`         DECIMAL(38,0) DEFAULT NULL,
INTERVAL_YEAR  VARCHAR(30) DEFAULT NULL,
INTERVAL_DAY   VARCHAR(30) DEFAULT NULL,
`LONG`        LONGTEXT DEFAULT NULL,
NCHAR          CHAR(10) DEFAULT NULL,
NCHAR_VARYING  VARCHAR(10) DEFAULT NULL,
NUMBER         DECIMAL(9,9) DEFAULT NULL,
NUMBER1        DECIMAL(9,0) DEFAULT NULL,
`NUMBER(*)`    DOUBLE DEFAULT NULL,
`NUMERIC`     DECIMAL(9,9) DEFAULT NULL,
NVARCHAR2      VARCHAR(10) DEFAULT NULL,
RAW            VARBINARY(10) DEFAULT NULL,
`REAL`        DOUBLE DEFAULT NULL,
ROW_ID         CHAR(10) DEFAULT NULL,
`SMALLINT`    DECIMAL(38,0) DEFAULT NULL,
`TIMESTAMP`   DATETIME(5) DEFAULT NULL,

```

```
TIMESTAMP_WITH_TIME_ZONE DATETIME(5) DEFAULT NULL,
UROWID VARCHAR(10) DEFAULT NULL,
`VARCHAR` VARCHAR(10) DEFAULT NULL,
VARCHAR2 VARCHAR(10) DEFAULT NULL,
XMLTYPE LONGTEXT DEFAULT NULL);
```

While most of the data types were converted successfully, a few exceptions were raised for data types that SCT is unable to automatically convert, and where SCT recommended manual actions:

- **MySQL does not have a data type BFILE:** BFILEs are pointers to binary files.

Recommended Actions: Either store a named file with the data and create a routine that gets that file from the file system, or store the data blob inside your database.

- **MySQL doesn't have a data type ROWID:** ROWIDs are physical row addresses inside Oracle's storage subsystems. The ROWID datatype is primarily used for values returned by the ROWID pseudocolumn.

Recommended Actions: Although MySQL contains a ctid column that is the physical location of the row version within its table, it does not have a comparable data type. However, you can use CHAR as a partial datatype equivalent. Note: If you are using ROWID datatypes in your code, modifications may be necessary.



- **MySQL does not have a data type UROWID:** Universal rowid, or UROWID, is a single Oracle datatype that supports both logical and physical rowids of foreign table rowids such as non-Oracle tables accessed through a gateway.

Recommended Actions: MySQL does not have a comparable data type. You can use VARCHAR(n) as a partial datatype equivalent. However, if you are using UROWID datatypes in your code, modifications may be necessary.

For more details, see:

- [AWS Schema Conversion Tool Documentation](#)
- <https://dev.mysql.com/doc/refman/5.7/en/data-types.html>

Oracle Read-Only Tables and Partitions vs. MySQL Aurora Replicas

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	READ ONLY mode is not supported, use a work-around

Oracle Usage

Beginning with Oracle 11g, tables can be marked as read-only to prevent DML operations from altering table data.

Prior to Oracle 11g, the only way to set a table to read-only mode was by limiting table privileges to SELECT. The table owner was still able to perform read and write operations. Beginning with Oracle 11g, users can execute an ALTER TABLE statement and change the table mode to either READ ONLY or READ WRITE.

Oracle 12c Release 2 introduces greater granularity for read-only objects and supports read-only table partitions. Any attempt to perform a DML operation on a partition, or sub-partition, set to READ ONLY results in an error.

Notes

- SELECT FOR UPDATE statements are not allowed.
- DDL operations are permitted if they do not modify table data.
- Operations on indexes are allowed on tables set to READ ONLY mode.

Examples

```
CREATE TABLE EMP_READ_ONLY (
EMP_ID          NUMBER PRIMARY KEY,
EMP_FULL_NAME  VARCHAR2(60) NOT NULL);

INSERT INTO EMP_READ_ONLY VALUES(1, 'John Smith');

1 row created

ALTER TABLE EMP_READ_ONLY READ ONLY;

INSERT INTO EMP_READ_ONLY VALUES(2, 'Steven King');

ORA-12081: update operation not allowed on table "SCT"."TBL_READ_ONLY"

ALTER TABLE EMP_READ_ONLY READ WRITE;

INSERT INTO EMP_READ_ONLY VALUES(2, 'Steven King');

1 row created

COMMIT;
```

```
SELECT * FROM EMP_READ_ONLY;
```

```
EMP_ID EMP_FULL_NAME
```

```
-----
```

```
1 John Smith
```

```
2 Steven King
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/ALTER-TABLE.html>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/vldbg/release-changes.html#GUID-C7A9BAD4-E4C9-4765-88C5-51AC7E97BAF1>

MySQL Usage

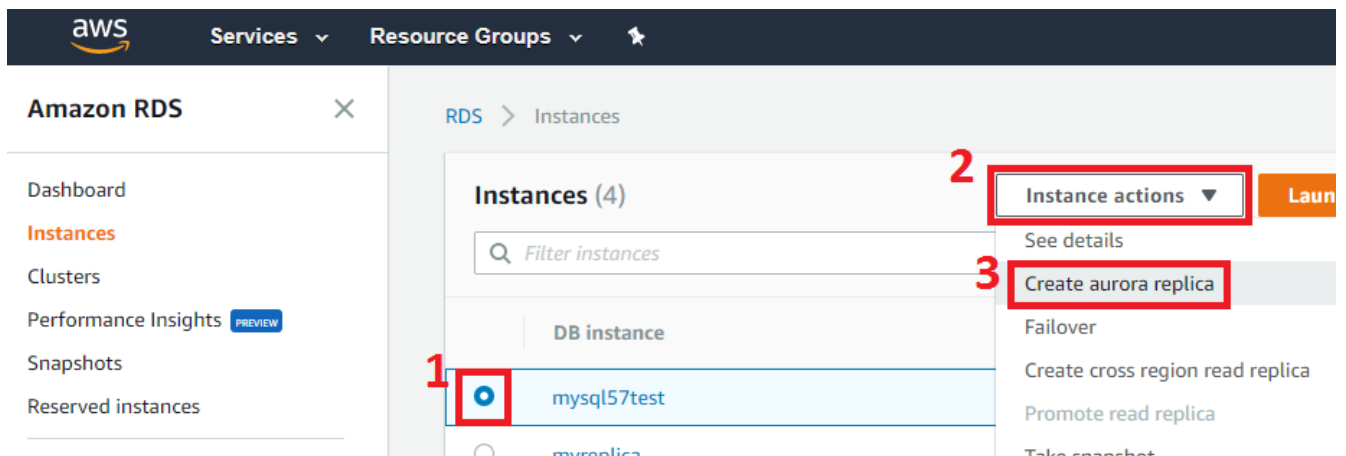
MySQL Aurora doesn't provide a built-in feature for read only tables, but the same functionality can be achieved using Aurora Replicas. The main disadvantage of this approach is that you must use two separated instances.

Note: It is important to note that there is a granularity difference between this workaround and options with Oracle. you cannot mimic a single read-only table, this workaround creates a read-only copy of the database.

Examples

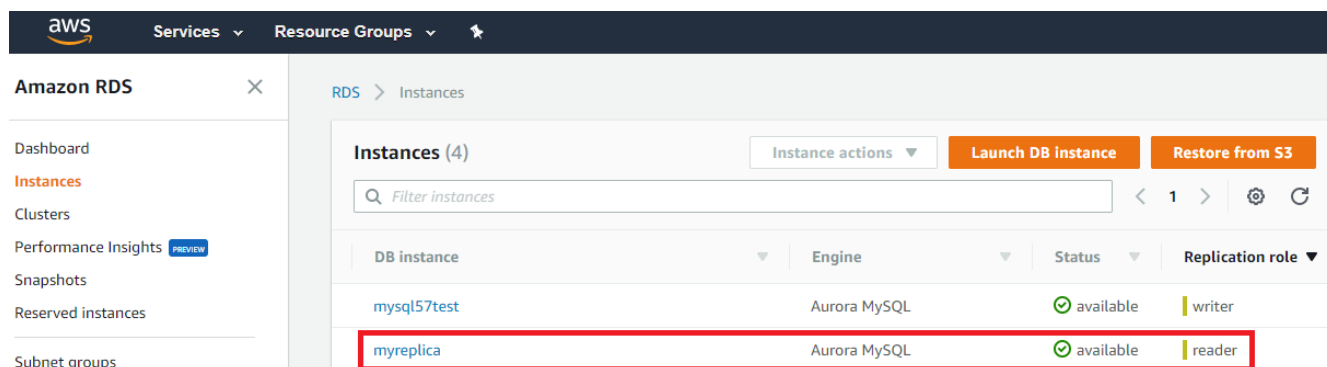
The following walkthrough demonstrates how to create an Aurora replica:

1. Locate the Aurora MySQL instance on the [instances page](#).
2. Select **Instance actions** and select **Create aurora replica**.



3. On the next page, enter the details and click **Create aurora replica**.

- View the new record on the [instances page](#) and wait until the **Status** changes to **available** and the **Replication role** changes to **reader**.





The screenshot displays the AWS Management Console interface for Amazon RDS. The left sidebar shows the navigation menu with 'Instances' selected. The main content area shows a list of instances. The instance 'myreplica' is highlighted with a red border, indicating it is the focus of the step. The table below shows the details of the instances.

DB instance	Engine	Status	Replication role
mysql57test	Aurora MySQL	available	writer
myreplica	Aurora MySQL	available	reader

For more details, see:

[Create an Amazon Aurora Read Replica from an RDS MySQL DB Instance](#)

Oracle Table Constraints vs. MySQL Table Constraints

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Constraints	CHECK, REF, DEFERRABLE, DISABLE constraints are not supported in MySQL Constraints on Views are not supported in MySQL

Oracle Usage

Oracle provides six types of constraints to enforce data integrity on table columns. Constraints ensure data inserted into tables is controlled and satisfies logical requirements.

Oracle Integrity Constraint Types

- **Primary Key:** Enforces that row values in a specific column are unique and not null.
- **Foreign Key:** Enforces that values in the current table exist in the referenced table.
- **Unique:** Prevents data duplication on a column, or combination of columns, and allows one null value.
- **Check:** Enforces that values comply with a specific condition.
- **Not Null:** Enforces that null values cannot be inserted into a specific column.
- **REF:** References an object in another object type or in a relational table.

Constraint Creation

You can create new constraints in two ways:

- **Inline:** Defines a constraint as part of a table column declaration.

```
CREATE TABLE EMPLOYEES (
    EMP_ID NUMBER PRIMARY KEY, ...);
```

- **Out-Of-Line:** Defines a constraint as part of the table DDL during table creation.

```
CREATE TABLE EMPLOYEES (EMP_ID NUMBER, ...,
    CONSTRAINT PK_EMP_ID PRIMARY KEY(EMP_ID));
```

Note: NOT NULL constraints must be declared using the inline method.

Oracle constraints can be specified with the following syntax:

- CREATE / ALTER TABLE
- CREATE / ALTER VIEW

Note: Views have only a primary key, foreign key, and unique constraints.

Privileges

You must have privileges on the table where constraints are created and, in case of foreign key constraints, you must have the REFERENCES privilege on the referenced table.

PRIMARY KEY Constraints

A unique identifier for each record in a database table can appear only once and cannot contain NULL values. A table can only have one primary key.

When creating a primary key constraint inline, you can specify only the PRIMARY KEY keyword. When you create the constraint out-of-line, you must specify one column or a combination of columns.

Creating a new primary key constraint also implicitly creates a unique index on the primary key column if no index already exists. When dropping a primary key constraint, the system-generated index is also dropped. If a user defined index was used, the index is not dropped.

- Primary keys cannot be created on columns defined with the following data types: LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE.

Note: The data type TIMESTAMP WITH LOCAL TIME ZONE is allowed as a primary key.

- Primary keys can be created from multiple columns (composite PK). They are limited to a total of 32 columns.
- Defining the same column as both a primary key and as a unique constraint is not allowed.

Examples

Create an Inline primary key using a system-generated primary key constraint name.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMBER PRIMARY KEY,
    FIRST_NAME  VARCHAR2 (20) ,
    LAST_NAME   VARCHAR2 (25) ,
    EMAIL       VARCHAR2 (25) );
```

Create an inline primary key using a user-specified primary key constraint name.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMBER CONSTRAINT PK_EMP_ID PRIMARY KEY,
    FIRST_NAME  VARCHAR2 (20) ,
    LAST_NAME   VARCHAR2 (25) ,
    EMAIL       VARCHAR2 (25) );
```

Create an out-of-line primary key.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMBER,
    FIRST_NAME  VARCHAR2 (20) ,
    LAST_NAME   VARCHAR2 (25) ,
    EMAIL       VARCHAR2 (25) );
CONSTRAINT PK_EMP_ID PRIMARY KEY (EMPLOYEE_ID);
```

Add a primary key to an existing table.

```
ALTER TABLE SYSTEM_EVENTS
  ADD CONSTRAINT PK_EMP_ID PRIMARY KEY (EVENT_CODE, EVENT_TIME);
```

FOREIGN KEY Constraint

Foreign key constraints identify the relationship between column records defined with a foreign key constraint and a referenced primary key or a unique column. The main purpose of a foreign key is to enforce that the values in table A also exist in table B as referenced by the foreign key.

A referenced table is known as a parent table. The table on which the foreign key was created is known as a child table. Foreign keys created in child tables generally reference a primary key constraint in a parent table.

Limitations

- Foreign keys cannot be created on columns defined with the following data types: LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE.
- Composite Foreign key constraints comprised from multiple columns cannot have more than 32 columns.
- Foreign key constraints cannot be created in a CREATE TABLE statement with a subquery clause.
- A referenced primary key or unique constraint on a parent table must be created before the foreign key creation command.

ON DELETE Clause

The ON DELETE clause specifies the effect of deleting values from a parent table on the referenced records of a child table. If the ON DELETE clause is not specified, Oracle does not allow deletion of referenced key values in a parent table that has dependent rows in the child table.

- **ON DELETE CASCADE:** Dependent foreign key values in a child table are removed along with the referenced values from the parent table.
- **ON DELETE NULL:** Dependent foreign key values in a child table are updated to NULL.

Examples

Create an inline foreign key with a user-defined constraint name.

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID    NUMBER PRIMARY KEY,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  DEPARTMENT_ID REFERENCES DEPARTMENTS (DEPARTMENT_ID));
```

Create an Out-Of-Line foreign key with a system-generated constraint name.

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID    NUMBER PRIMARY KEY,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  DEPARTMENT_ID NUMBER,
```

```
CONSTRAINT FK_FEP_ID
FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS (DEPARTMENT_ID));
```

Create a foreign key using the ON DELETE CASCADE clause.

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID NUMBER PRIMARY KEY,
  FIRST_NAME VARCHAR2(20),
  LAST_NAME VARCHAR2(25),
  EMAIL VARCHAR2(25),
  DEPARTMENT_ID NUMBER,
  CONSTRAINT FK_FEP_ID
  FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS (DEPARTMENT_ID)
  ON DELETE CASCADE);
```

Add a foreign key to an existing table:

```
ALTER TABLE EMPLOYEES
  ADD CONSTRAINT FK_FEP_ID
  FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS (DEPARTMENT_ID);
```

UNIQUE Constraints

A unique constraint is similar to a primary key constraint. It specifies that the values in a single column, or combination of columns, must be unique and cannot repeat in multiple rows.

The main difference from primary key constraint is that a unique constraint can contain NULL values. NULL values in multiple rows are also supported provided the combination of values is unique.

Limitations

- A unique constraint cannot be created on columns defined with the following data types: LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE
- A unique constraint comprised from multiple columns cannot have more than 32 columns.
- Primary key and unique constraints cannot be created on the same column or columns.

Examples

Create an inline unique Constraint:

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID NUMBER PRIMARY KEY,
  FIRST_NAME VARCHAR2(20),
  LAST_NAME VARCHAR2(25),
  EMAIL VARCHAR2(25) CONSTRAINT UNIQ_EMP_EMAIL UNIQUE,
  DEPARTMENT_ID NUMBER);
```

Check Constraints

Check constraints are used to validate that values in specific columns meet specific criteria or conditions. For example, a check constraint on an EMPLOYEE_EMAIL column can be used to validate that each record has an

“@aws.com” suffix. If a record fails the “check” validation, an error is raised and the record is not inserted.

Using a check constraint can help transfer some of the logical integrity validation from the application to the database.

In-Line vs. Out-Of-Line

When creating a check constraint as inline, it can only be defined on a specific column. When using the out-of-line method, the check constraint can be defined on multiple columns.

Limitations

- Check constraints cannot perform validation on columns of other tables.
- Check constraints cannot be used with functions that are not deterministic (e.g. CURRENT_DATE).
- Check constraints cannot be used with user-defined functions.
- Check constraints cannot be used with pseudo columns such as: CURRVAL, NEXTVAL, LEVEL, or ROWNUM.

Examples

Create an inline check constraint that uses a regular expression to validate the email suffix of inserted rows contains “@aws.com”.

```
CREATE TABLE EMPLOYEES (  
    EMPLOYEE_ID    NUMBER PRIMARY KEY,  
    FIRST_NAME     VARCHAR2(20),  
    LAST_NAME      VARCHAR2(25),  
    EMAIL          VARCHAR2(25)  
    CHECK (REGEXP_LIKE (EMAIL, '^ [A-Za-z]+@aws.com?{1,3}$')),  
    DEPARTMENT_ID NUMBER);
```

Not Null Constraints

A Not Null constraint prevent a column from containing any null values. To enable the not null constraint, the keywords NOT NULL must be specified during table creation (inline only). Permitting null values is the default if NOT NULL is not specified.

Examples

```
CREATE TABLE EMPLOYEES (  
    EMPLOYEE_ID    NUMBER PRIMARY KEY,  
    FIRST_NAME     VARCHAR2(20) NOT NULL,  
    LAST_NAME      VARCHAR2(25) NOT NULL,  
    EMAIL          VARCHAR2(25),  
    DEPARTMENT_ID NUMBER);
```

REF Constraints

REF constraints define a relationship between a column of type REF and the object it references. The REF constraint can be created both inline and out-of-line. Both methods permit defining a scope constraint, a rowid constraint, or a referential integrity constraint based on the REF column.

Examples

Create a new Oracle type object.

```
CREATE TYPE DEP_TYPE AS OBJECT (
    DEP_NAME    VARCHAR2(60),
    DEP_ADDRESS VARCHAR2(300));
```

Create a table based on the previously created type object.

```
CREATE TABLE DEPARTMENTS_OBJ_T OF DEP_TYPE;
```

Create the EMPLOYEES table with a reference to the previously created DEPARTMENTS table that is based on the DEP_TYPE object:

```
CREATE TABLE EMPLOYEES (
    EMP_NAME    VARCHAR2(60),
    EMP_EMAIL   VARCHAR2(60),
    EMP_DEPT    REF DEPARTMENT_TYP REFERENCES DEPARTMENTS_OBJ_T);
```

Special Constraint States

Oracle provides granular control of database constraint enforcement. For example, you can disable constraints temporarily while making modifications to table data.

Constraint states can be defined using the CREATE TABLE / ALTER TABLE statements. The following constraint states are supported:

- **DEFERRABLE**: Enables the use of the SET CONSTRAINT clause in subsequent transactions until a COMMIT statement is submitted.
- **NOT DEFERRABLE**: Disables the use of the SET CONSTRAINT clause.
- **INITIALLY IMMEDIATE**: Checks the constraint at the end of each subsequent SQL statement (this state is the default).
- **INITIALLY DEFERRED**: Checks the constraint at the end of subsequent transactions.
- **VALIDATE | NO VALIDATE**: These parameters depend on whether the constraint is ENABLED or DISABLED.
- **ENABLE | DISABLE**: Specifies if the constraint should be enforced after creation (ENABLE by default). Several options are available when using ENABLE | DISABLE:
 - **ENABLE VALIDATE**: Enforces that the constraint applies to all existing and new data.
 - **ENABLE NOVALIDATE**: Only new data complies with the constraint.
 - **DISABLE VALIDATE**: A valid constraint is created in disabled mode with no index.

- **DISABLE NOVALIDATE:** The constraint is created in disabled mode without validation of new or existing data.

Examples

Create a unique constraint with a state of DEFERRABLE.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMBER PRIMARY KEY,
    FIRST_NAME VARCHAR2(20),
    LAST_NAME VARCHAR2(25),
    EMAIL VARCHAR2(25) CONSTRAINT UNIQ_EMP_EMAIL
UNIQUE DEFERRABLE,
    DEPARTMENT_ID NUMBER);
```

Modify the state of the constraint to ENABLE NOVALIDATE.

```
ALTER TABLE EMPLOYEES
    ADD CONSTRAINT CHK_EMP_NAME CHECK(FIRST_NAME LIKE 'a%')
    ENABLE NOVALIDATE;
```

Using Existing Indexes to Enforce Constraint Integrity (using_index_clause)

Primary key and unique constraints can be created based on an existing index to enforce the constraint integrity instead of implicitly creating a new index during constraint creation.

Examples

Create a unique constraint based on an existing index.

```
CREATE UNIQUE INDEX IDX_EMP_ID ON EMPLOYEES(EMPLOYEE_ID);

ALTER TABLE EMPLOYEES
    ADD CONSTRAINT PK_CON_UNIQ
    PRIMARY KEY(EMPLOYEE_ID) USING INDEX IDX_EMP_ID;
```

For more details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.html#GUID-F9CE0CC3-13AE-4744-A43C-EAC7A71AAAB6>

MySQL Usage

MySQL supports the following types of table constraints:

- PRIMARY KEY
- FOREIGN KEY
- UNIQUE

- NOT NULL
- ENUM (unique to MySQL)
- SET (unique to MySQL)

Note: MySQL does not support Oracle REF constraint.

Similar to constraint deceleration in Oracle, MySQL allows creating constraints inline or out-of-line when specifying table columns.

MySQL constraints can be specified using CREATE / ALTER TABLE. Views are not supported.

Privileges

You must have privileges on the table in which constrains are created. For foreign key constraints, you must also have the REFERENCES privilege.

Primary Key Constraints

- Marks the column(s) on which the table's heap is sorted (in the InnoDB storage engine, like Oracle IOT).
- Uniquely identifies each record and cannot contain a NULL value.
- Uses the same ANSI SQL syntax as Oracle.
- Can be created on a single column, or on multiple columns (composite primary keys), as the only PRIMARY KEY in a table.
- Creates a unique B-Tree index automatically on the column, or group of columns, marked as the primary key of the table.
- Constraint names can be generated automatically by MySQL. If a name is explicitly specified during constraint creation, the constraint name is still be PRIMARY.

Examples

Create an inline primary key constraint.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMERIC PRIMARY KEY,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(25),
    EMAIL VARCHAR(25));
```

Create an out-of-line primary key constraint (note that in both examples the constraint name is PRIMARY).

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMERIC,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(25),
    EMAIL VARCHAR(25),
    CONSTRAINT PK_EMP_ID PRIMARY KEY (EMPLOYEE_ID));
OR
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMERIC,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(25),
```

```
EMAIL          VARCHAR(25),
CONSTRAINT PRIMARY KEY (EMPLOYEE_ID);
```

Add a primary key constraint to an existing table (all the examples below produce the same result).

```
ALTER TABLE SYSTEM_EVENTS
    ADD CONSTRAINT PK_EMP_ID PRIMARY KEY (EVENT_CODE, EVENT_TIME);

OR

ALTER TABLE SYSTEM_EVENTS
    ADD CONSTRAINT PRIMARY KEY (EVENT_CODE, EVENT_TIME);

OR

ALTER TABLE SYSTEM_EVENTS
    ADD PRIMARY KEY (EVENT_CODE, EVENT_TIME);
```

Drop the primary key.

```
ALTER TABLE SYSTEM_EVENTS DROP PRIMARY KEY;
```

Foreign Key Constraints

Important notes about foreign key constraints:

- Enforces referential integrity in the database. Values in specific columns or group of columns must match the values from another table (or column).
- Creating a FOREIGN KEY constraint in MySQL uses the same ANSI SQL syntax as Oracle.
- Can be created only out-of-line during table creation.
- Use the REFERENCES clause to specify the table referenced by the foreign key constraint.
- A table can have multiple FOREIGN KEY constraints to describe its relationships with other tables.
- Use the ON DELETE clause to handle cases of FOREIGN KEY parent records deletions (such as cascading deletes).
- Use the ON UPDATE clause to handle cases of FOREIGN KEY parent records updates (such as cascading updates).
- Foreign key constraint names are generated automatically by the database or specified explicitly during constraint creation.

ON DELETE clause

MySQL provides four options to handle cases where data is deleted from the parent table and a child table is referenced by a FOREIGN KEY constraint. By default, without specifying any additional options, MySQL uses the NO ACTION method and raises an error if the referencing rows still exist when the constraint is verified.

- **ON DELETE CASCADE:** Removes any dependent foreign key values in the child table along with the referenced values from the parent table.
- **ON DELETE RESTRICT:** Prevents the deletion of referenced values from the parent table and the deletion of dependent foreign key values in the child table.
- **ON DELETE NO ACTION:** Prevents the deletion of referenced values from the parent table and the deletion of dependent foreign key values in the child table (the same as RESTRICT).

- **ON DELETE SET NULL:** Deletes the row from the parent table and sets the foreign key column, or columns in the child table, to NULL. If you specify a SET NULL action, ensure you have not declared the columns in the child table as NOT NULL.

ON UPDATE clause

Handle updates on FOREIGN KEY columns using the ON UPDATE clause, which shares the same options as the ON DELETE clause:

- ON UPDATE CASCADE
- ON UPDATE RESTRICT
- ON UPDATE NO ACTION

Note: Oracle does not provide an ON UPDATE clause.

Examples

Create an out-of-line foreign key constraint with a system-generated constraint name.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMERIC PRIMARY KEY,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(25),
    EMAIL VARCHAR(25),
    DEPARTMENT_ID NUMERIC,
    CONSTRAINT FK_FEP_ID FOREIGN KEY (DEPARTMENT_ID)
        REFERENCES DEPARTMENTS (DEPARTMENT_ID));
```

Create a foreign key using the ON DELETE CASCADE clause.

```
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMERIC PRIMARY KEY,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(25),
    EMAIL VARCHAR(25),
    DEPARTMENT_ID NUMERIC,
    CONSTRAINT FK_FEP_ID FOREIGN KEY (DEPARTMENT_ID)
        REFERENCES DEPARTMENTS (DEPARTMENT_ID) ON DELETE CASCADE);
```

Add a foreign key to an existing table.

```
ALTER TABLE EMPLOYEES ADD CONSTRAINT FK_FEP_ID
    FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS (DEPARTMENT_ID);
```

UNIQUE Constraints

Important notes about unique constraints:

- Ensures that a value in a column, or a group of columns, is unique across the entire table.
- MySQL UNIQUE constraint syntax is ANSI SQL compatible.
- Automatically creates a B-Tree index on the respective column, or a group of columns, when creating a UNIQUE constraint.

- If duplicate values exist in the column(s) on which the constraint was defined during UNIQUE constraint creation, the UNIQUE constraint creation fails and returns an error message.
- UNIQUE constraints in MySQL accept multiple NULL values (similar to Oracle).
- UNIQUE constraint naming can be system-generated or explicitly specified.

Examples

Create an inline unique constraint ensuring uniqueness of values in the email column.

```
CREATE TABLE EMPLOYEES (
EMPLOYEE_ID    NUMERIC PRIMARY KEY,
FIRST_NAME     VARCHAR(20),
LAST_NAME      VARCHAR(25),
EMAIL          VARCHAR(25) UNIQUE,
DEPARTMENT_ID NUMERIC);
```

Note: NOT NULL does not create a database object and does not create a reference in the constraints metadata table.

Disable Integration Check

MySQL has no option to DISABLE the integration check, but there is a session variable for disabling checks at the session level.

Turn on integration checks in the session.

```
SET FOREIGN_KEY_CHECKS=1;
```

Turn off the integration checks in the session.

```
SET FOREIGN_KEY_CHECKS=0;
```

Constraints Unique to MySQL

- **ENUM:** The value must be one of the values listed in the column definition or the internal numeric equivalent. The value cannot be the error value (that is, 0 or the empty string). For a column defined as ENUM ('a','b','c'), values such as "", 'd', or 'ax' are invalid and are rejected.
- **SET:** The value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as SET('a','b','c'), values such as 'd' or 'a,b,c,d' are invalid and are rejected.

Summary



Oracle Constraint / Parameter	MySQL Constraint / Parameter
PRIMARY KEY	PRIMARY KEY
FOREIGN KEY	FOREIGN KEY
UNIQUE	UNIQUE
CHECK	Not supported, in some cases ENUM and SET can be used. Can be imple-

Oracle Constraint / Parameter	MySQL Constraint / Parameter
	mented with triggers
NOT NULL	NOT NULL
REF	Not supported
DEFERRABLE	Not supported as keyword, can use the FOREIGN_KEY_CHECKS parameter.
NOT DEFERRABLE	Not supported as keyword, can use the FOREIGN_KEY_CHECKS parameter.
SET CONSTRAINTS	Not supported as keyword, can use the FOREIGN_KEY_CHECKS parameter.
INITIALLY IMMEDIATE	Default, not supported as keyword.
INITIALLY DEFERRED	Not supported.
ENABLE	Default, not supported as keyword.
DISABLE	Not supported as keyword, can use the FOREIGN_KEY_CHECKS parameter.
ENABLE VALIDATE	Default, not supported as keyword.
ENABLE NOVALIDATE	Not supported.
DISABLE VALIDATE	Not supported.
DISABLE NOVALIDATE	Default, not supported as keyword.
USING_INDEX_CLAUSE	Not supported.
View Constraints	Not supported.
Metadata: DBA_CONSTRAINTS	Metadata: TABLE_CONSTRAINTS.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/constraints.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-table-foreign-keys.html>

Oracle Temporary Tables vs. MySQL Temporary Tables

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Temporary Table	<p>GLOBAL temporary table is not supported in MySQL</p> <p>Can't read from multiple sessions in MySQL</p> <p>Table dropped after session ends in MySQL</p>

Oracle Usage

In Oracle, you can create temporary tables for storing data that exists only for the duration of a session or transaction.

Use the CREATE GLOBAL TEMPORARY TABLE statement to create a temporary table. This type of table has a persistent DDL structure, but not persistent data. It does not generate redo during DML. Two of the primary use-cases for temporary tables include:

- Processing many rows as part of a batch operation while requiring staging tables to store intermediate results.
- Storing data required only for the duration of a specific session. When the session ends, the session data is cleared.

When using temporary tables, the data is visible only to the session that inserts the data into the table.

Oracle 18c introduces private temporary tables which are temporary tables that are only available during session or transaction. After session or transaction ends they are automatically dropped.

Oracle Global Temporary Tables

- Global Temporary Tables store data in the Oracle Temporary Tablespace.
- DDL operations on a temporary table are permitted including ALTER TABLE, DROP TABLE, and CREATE INDEX.
- Temporary tables cannot be partitioned, clustered, or created as Index-Organized Tables. Also, they do not support parallel UPDATE, DELETE, and MERGE.
- Foreign key constraints cannot be created on temporary tables.
- Processing DML operations on a Temporary Table does not generate Redo Data. However, Undo Data for the rows and Redo Data for the Undo Data itself are generated.
- Indexes can be created for a Temporary Table. They are treated as Temporary Indexes. Temporary Tables also support Triggers.
- Temporary Tables cannot be named after an existing table object and cannot be dropped while containing records, even from another session.

Session-Specific and Transaction-Specific Temporary Table Syntax

ON COMMIT: Specifies whether the temporary table data persists for the duration of a transaction or a session.

- **PRESERVE ROWS:** When the session ends, all data is truncated but persists beyond the end of the transaction.
- **DELETE ROWS:** The default behavior. Data is truncated after each commit.

Oracle 12c Temporary Table Enhancements

Global Temporary Table Statistics

Prior to Oracle 12c, statistics on temporary tables were common to all sessions. Oracle 12c introduces session-specific statistics for Temporary Tables. Statistics can be configured using the DBMS_STATS preference GLOBAL_TEMP_TABLE_STATS, which can be set to SHARED or SESSION.

Global Temporary Table Undo

Performing DML operations on a Temporary Table does not generate Redo data, but does generate Undo Data that eventually, by itself, generates Redo records. Oracle 12c provides an option to store the temporary Undo Data in the Temporary Tablespace itself. This feature is configured using the temp_undo_enabled parameter with the options TRUE or FALSE.

For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/refm/TEMP_UNDO_ENABLED.htm-#GUID-E2A01A84-2D63-401F-B64E-C96B18C5DCA6

Examples

Create an Oracle Global Temporary Table (with ON COMMIT PRESERVE ROWS).

```
CREATE GLOBAL TEMPORARY TABLE EMP_TEMP (
    EMP_ID          NUMBER PRIMARY KEY,
    EMP_FULL_NAME   VARCHAR2(60) NOT NULL,
    AVG_SALARY      NUMERIC NOT NULL)
ON COMMIT PRESERVE ROWS;

CREATE INDEX IDX_EMP_TEMP_FN ON EMP_TEMP(EMP_FULL_NAME);

INSERT INTO EMP_TEMP VALUES(1, 'John Smith', '5000');

COMMIT;

SELECT * FROM SCT.EMP_TEMP;

EMP_ID EMP_FULL_NAME          AVG_SALARY
-----
1 John Smith                  5000
```

Create an Oracle Global Temporary Table (with ON COMMIT DELETE ROWS).

```
CREATE GLOBAL TEMPORARY TABLE EMP_TEMP (
  EMP_ID          NUMBER PRIMARY KEY,
  EMP_FULL_NAME  VARCHAR2(60) NOT NULL,
  AVG_SALARY     NUMERIC NOT NULL)
ON COMMIT DELETE ROWS;

INSERT INTO EMP_TEMP VALUES(1, 'John Smith', '5000');

COMMIT;

SELECT * FROM SCT.EMP_TEMP;
```

For more details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.html#GUID-F9CE0CC3-13AE-4744-A43C-EAC7A71AAAB6>

MySQL Usage

MySQL Temporary Tables share many similarities with Oracle Global Temporary Tables. From a syntax perspective, MySQL Temporary Tables are referred to as “Temporary Tables” (without Oracle’s Global definition). The implementation is mostly identical.

In terms of differences, Oracle stores the temporary table structure (DDL) for repeated use – even after a database restart – but does not store rows persistently. MySQL implements temporary tables differently. The table structure (DDL) is not stored in the database. When a session ends, the temporary table is dropped.

- **Session-Specific:** In MySQL, each session is required to create its own Temporary Tables. Each session can create its own “private” Temporary Tables using identical table names.
- In Oracle, the default behavior when the ON COMMIT clause is omitted is ON COMMIT DELETE ROWS. In MySQL, the default is ON COMMIT PRESERVE ROWS and it cannot be changed.

RDS MySQL Only (not Aurora MySQL): As of MySQL 8.0.13, user-created temporary tables and internal temporary tables created by the optimizer are stored in session temporary tablespaces that are allocated to a session from a pool of temporary tablespaces. When a session disconnects its temporary tablespaces are truncated and released back to the pool. In previous releases temporary tables were created in the global temporary tablespace (ibtmp1) which did not return disk space to the operating system after temporary tables were dropped. The `innodb_temp_tablespaces_dir` variable defines the location where session temporary tablespaces are created. The default location is the `#innodb_temp` directory in the data directory. The `INNODB_SESSION_TEMP_TABLESPACES` table provides metadata about session temporary tablespaces. The global temporary tablespace (ibtmp1) now stores rollback segments for changes made to user-created temporary tables.

Examples

```
CREATE TEMPORARY TABLE EMP_TEMP (
  EMP_ID          INT PRIMARY KEY,
```



```
EMP_FULL_NAME VARCHAR(60) NOT NULL,  
AVG_SALARY     INT NOT NULL1;
```

Summary

	Oracle	Aurora MySQL
Semantic	Global Temporary Table	Temporary Table
Create table	CREATE GLOBAL TEMPORARY...	CREATE TEMPORARY...
Accessible from multiple sessions	Yes	No
Temp table DDL persist after session end / database restart user-managed datafiles	Yes	No (dropped at the end of the session)
Create index support	Yes	Yes
Foreign key support	Yes	No
ON COMMIT default	COMMIT DELETE ROWS	ON COMMIT PRESERVE ROWS
ON COMMIT PRESERVE ROWS	Yes	No
ON COMMIT DELETE ROWS	Yes	No
Alter table support	Yes	Yes
Gather statistics	dbms_stats.gather_table_stats	ANALYZE
Oracle 12c GLOBAL_TEMP_TABLE_STATS	dbms_stats.set_table_prefs	ANALYZE

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/create-temporary-table.html>

Oracle Triggers vs. MySQL Triggers

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Triggers	Statement and System event triggers are not supported in MySQL CREATE OR REPLACE is not supported in MySQL

Oracle Usage

A trigger is a “named program” written in PL/SQL, stored in the database, and fired (or executed) when a specified event occurs. The associated event causing a trigger to execute can either be tied to a specific database table, database view, database schema, or the database itself.

Triggers can be executed after:

- Data Manipulation Language (DML) statements (DELETE, INSERT, or UPDATE).
- Data Definition Language (DDL) statements (CREATE, ALTER, or DROP).
- Certain database events and operations (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Trigger Types:

- **DML Trigger:** Can be created on Tables or Views and fire when inserting, updating, or deleting data. Triggers can fire before or after DML command execution.
- **INSTEAD OF Trigger:** A special type of DML trigger created on a non-editable view. INSTEAD OF triggers provide an application-transparent method for modifying views that cannot be modified via DML statements.
- **SYSTEM Event Triggers:** Triggers defined at the database or schema level including triggers that fire after specific events:
 - User log-on and log-off.
 - Database events (startup/shutdown), DataGuard events, server errors.

Examples

Create a trigger that is executed after a row is deleted from the PROJECTS table, or if the primary key of a project is updated.

```
CREATE OR REPLACE TRIGGER PROJECTS_SET_NULL
  AFTER DELETE OR UPDATE OF PROJECTNO ON PROJECTS
  FOR EACH ROW
  BEGIN
    IF UPDATING AND :OLD.PROJECTNO != :NEW.PROJECTNO OR DELETING THEN
      UPDATE EMP SET EMP.PROJECTNO = NULL
        WHERE EMP.PROJECTNO = :OLD.PROJECTNO;
    END IF;
```

```

END;
/

Trigger created.

DELETE FROM PROJECTS WHERE PROJECTNO=123;

SELECT PROJECTNO FROM EMP WHERE PROJECTNO=123;

PROJECTNO
-----
NULL

```

Create a SYSTEM/Schema trigger on a table. The trigger fires if a DDL DROP command is executed for an object in the HR schema. It prevents dropping the object and raises an application error.

```

CREATE OR REPLACE TRIGGER PREVENT_DROP_TRIGGER
  BEFORE DROP ON HR.SCHEMA
  BEGIN
    RAISE_APPLICATION_ERROR (num => -20000,
      msg => 'Cannot drop object');
END;
/

Trigger created.

DROP TABLE HR.EMP

ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20000: Cannot drop object
ORA-06512: at line 2

```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/CREATE-TRIGGER-statement.html#GUID-AF9E33F1-64D1-4382-A6A4-EC33C36F237B>

MySQL Usage

MySQL supports triggers, but not all of the functionality provided by Oracle. Triggers are associated with users (for privileges reasons), are associated with specific tables, and fire at the row level (not the statement level). You can modify MySQL triggers using a FOLLOWS or PRECEDES clause. Also, MySQL triggers can be chained using the FOLLOWS or PRECEDES clauses.

CREATE TRIGGER Syntax

```

CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
[trigger_order]
trigger_body

```

```
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Examples

Create a trigger referencing the OLD and NEW values.

```
set delimiter /
CREATE OR REPLACE TRIGGER PROJECTS_SET_NULL
  BEFORE UPDATE ON PROJECTS
  FOR EACH ROW
  BEGIN
    IF OLD.PROJECTNO != NEW.PROJECTNO THEN
      UPDATE EMP SET EMP.PROJECTNO = NULL
        WHERE EMP.PROJECTNO = OLD.PROJECTNO;
    END IF;
  END;
/
set delimiter ;

UPDATE PROJECTS WHERE PROJECTNO=123;

SELECT PROJECTNO FROM EMP WHERE PROJECTNO=123;

PROJECTNO
-----
NULL
```

Drop a trigger.

```
DROP TRIGGER PROJECTS_SET_NULL
```

Summary


	Oracle	Aurora MySQL
“Before update” trigger, row level	<pre>CREATE OR REPLACE TRIGGER check_update BEFORE UPDATE ON projects FOR EACH ROW BEGIN /*Trigger body*/ END; /</pre>	<pre>CREATE TRIGGER check_update BEFORE UPDATE ON projects FOR EACH ROW BEGIN /*Trigger body*/ END; /</pre>
“Before update” trigger, statement level	<pre>CREATE OR REPLACE TRIGGER check_update BEFORE UPDATE ON projects BEGIN /*Trigger body*/ END;</pre>	Not Supported

	Oracle	Aurora MySQL
	/	
System / event trigger	<pre>CREATE OR REPLACE TRIGGER drop_trigger BEFORE DROP ON hr.SCHEMA BEGIN RAISE_APPLICATION_ERROR (num => -20000, msg => 'Cannot drop object'); END; /</pre>	Not supported
Referencing :old and :new values in triggers	<p>Use ":NEW" and ":OLD" in trigger body:</p> <pre>CREATE OR REPLACE TRIGGER UpperNewDeleteOld BEFORE INSERT OR UPDATE OF first_name ON employees FOR EACH ROW BEGIN :NEW.first_name := UPPER(:NEW.first_name); :NEW.salary := :OLD.salary; END; /</pre>	<p>Use "NEW" and "OLD" in trigger body:</p> <pre>CREATE TRIGGER UpperNewDeleteOld BEFORE UPDATE ON empys FOR EACH ROW SET NEW.first_name = UPPER(NEW.first_name), NEW.salary = OLD.salary; END; /</pre>
Database event level trigger	<pre>CREATE TRIGGER register_shutdown ON DATABASE SHUTDOWN BEGIN Insert into logging values ('DB was shutdown', sysdate); commit; END; /</pre>	Not supported
Drop a trigger	DROP TRIGGER last_name_change_trg ;	DROP TRIGGER last_name_change_trg ;
Modify logic executed by a trigger	<p>Can be used with create or replace</p> <pre>CREATE OR REPLACE TRIGGER UpperNewDeleteOld BEFORE INSERT OR UPDATE OF first_name ON employees FOR EACH ROW BEGIN <<NEW CONTENT>> END; /</pre>	Not supported
Enable a trigger	ALTER TRIGGER UpperNewDeleteOld ENABLE;	<p>Not supported</p> <p>Can be achieved by setting variables for each trigger to determine if it is disabled or enabled, and then checking the variable in an IF statement.</p>
Disable a trigger	ALTER TRIGGER UpperNewDeleteOld DISABLE;	<p>Not supported</p> <p>Can be achieved as described above.</p>

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-trigger.html>

Oracle Tablespaces and Data Files vs. MySQL Tablespaces and Data files

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Tablespace-per-file only and no physical files attributes allowed in Aurora MySQL

Oracle Usage

The storage structure of an Oracle database contains both physical and logical elements.

- **Tablespaces:** Each Oracle database contains one or more tablespaces, which are logical storage groups used as “containers” for creating new tables and indexes.
- **Data Files:** Each tablespace is made up of one or more data files, which are the physical elements of an Oracle database tablespace. Datafiles can be located on the local file system, located in raw partitions, managed by Oracle ASM, or located on a network file system.

Storage Hierarchy

- **Database:** Each Oracle database is composed of one or more tablespaces.
- **Tablespace:** Each Oracle tablespace is composed of one or more datafiles. Tablespaces are logical entities that have no physical manifestation on the file system.
- **Data Files:** Physical files located on a file system. Each Oracle tablespace consists of one or more data files.
- **Segments:** Each segment represents a single database object that consumes storage such as tables, indexes, and undo segments.
- **Extent:** Each segment consists of one or more extents. Oracle uses extents to allocate contiguous sets of database blocks on disk.
- **Block:** The smallest unit of I/O for reads and writes. For blocks storing table data, each block can store one or more table rows.

Types of Oracle Database Tablespaces

- **Permanent Tablespaces:** Designated to store persistent schema objects for applications.
- **Undo Tablespace :** A special type of system permanent tablespace used by Oracle to manage UNDO data when running the database in automatic undo management mode.
- **Temporary Tablespace:** Contains schema objects valid for the duration of a session. It is also used running operations that cannot fit into memory.

Tablespace Privileges

The following criteria must be met to create a tablespace:

- The database user must have the CREATE TABLESPACE system privilege.
- The database must be in OPEN mode.

Examples

Create a USERS tablespace comprised of a single data file.

```
CREATE TABLESPACE USERS
  DATAFILE '/u01/app/oracle/oradata/orcl/users01.dbf' SIZE 5242880
  AUTOEXTEND ON NEXT 1310720 MAXSIZE 32767M
  LOGGING ONLINE PERMANENT BLOCKSIZE 8192
  EXTENT MANAGEMENT LOCAL AUTOALLOCATE DEFAULT
  NOCOMPRESS SEGMENT SPACE MANAGEMENT AUTO;
```

Drop a tablespace.

```
DROP TABLESPACE USERS;
OR
DROP TABLESPACE USERS INCLUDING CONTENTS AND DATAFILES;
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLESPACE.html#GUID-51F07BF5-EFAF-4910-9040-C473B86A8BF9>
- https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/file_specification.html#GUID-580FA726-F712-4410-90CF-783A2DA89688
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/DROP-TABLESPACE.html#GUID-C91F3E94-4503-48DE-9BCA-42E495E6BE11>

MySQL Usage

Aurora MySQL logical storage structure is similar to Oracle. It uses tablespaces for storing database objects, but the General Tablespace is not yet supported. Only InnoDB file-per-table is provided.

RDS MySQL Only (not Aurora MySQL): Starting with MySQL 8 renaming a general tablespace is supported by ALTER TABLESPACE ... RENAME TO syntax.

Terms:

- **Tablespace:** A logical unit to gather datafiles (in MySQL Aurora file-per-table tablespaces).
- **Data files:** File system files placed in a tablespace and used to store database objects such as tables and indexes. Data files are created automatically by MySQL, similar to the behavior of Oracle-Managed-Files (OMF).

The InnoDB file-per-table feature applies to each InnoDB table. Its indexes are stored in a separate .ibd data file. Each .ibd data file represents an individual tablespace.

Tablespaces

After an Amazon Aurora MySQL cluster is created, the tablespaces listed below are automatically provisioned and cannot be modified or dropped. These tablespaces hold database metadata or provide temporary storage for sorting and calculations :

- innodb_system
- innodb_temporary
- innodb_file_per_table_n

One of the main advantages of using Amazon Aurora MySQL is the reduced complexity of storage management. There is no need to create tablespaces because Aurora MySQL uses a unique, self-managed shared storage architecture. Database administrators do not need to micro-manage most storage aspects of databases.

Example

View all tablespaces.

```
SELECT * FROM INFORMATION_SCHEMA.FILES;
```

Summary



Feature	Oracle	Aurora MySQL
Tablespace	Exists as a logical object and consists of one or more user-specified or system-generated data files.	Exists as a logical object and consists of one data file.
Data file	<ul style="list-style-type: none"> • Can be explicitly created and resized by the user. Oracle-Managed-Files (OMF) allow for automatically created data files. • Each data file can contain one or more tables and/or indexes. 	Behavior is more like Oracle Managed Files (OMF): <ul style="list-style-type: none"> • Created automatically in the directory assigned to the tablespace. • A single data file stores information for a specific table or index. Multiple data files can exist for a table or index.
Creates a new tablespace with system-managed datafiles	CREATE TABLESPACE sales_tbs DATAFILE SIZE 400M;	N/A
Create a new tablespace with user-managed datafiles	CREATE TABLESPACE sales_tbs DATAFILE '/oradata/sales01.dbf' SIZE 1M AUTOEXTEND ON NEXT 1M;	N/A
Alter the size of a datafile	ALTER DATABASE DATAFILE '/oradata/-sales01.dbf' RESIZE 100M;	ALTER TABLE EMPLOYEES FORCE; Reclaims free space in the data file, which can reduce and tablespace size.
Add a datafile to an	ALTER TABLESPACE sales_tbs ADD	N/A

Feature	Oracle	Aurora MySQL
existing tablespace	DATAFILE '/oradata/sales02.dbf' SIZE 10M;	
Per-database tablespace	<p>Supported as part of the Oracle 12c Multi-Tenant architecture. Different dedicated tablespaces can be created for different pluggable databases and set as the default tablespace for a PDB:</p> <pre>ALTER SESSION SET CONTAINER = 'sales';</pre> <pre>CREATE TABLESPACE sales_tbs DATAFILE '/oradata/sales01.dbf' SIZE 1M AUTOEXTEND ON NEXT 1M;</pre> <pre>ALTER DATABASE sales TABLESPACE sales_tds;</pre>	N/A
Metadata tables	Data Dictionary tables are stored in the SYSTEM tablespace.	Data Dictionary tables are stored in the innodb_system tablespace.
Tablespace data encryption	<p>Supported</p> <ul style="list-style-type: none"> • Supported using transparent data encryption. • Encryption and decryption are handled seamlessly. Users do not have to modify the application to access the data. 	<p>Supported</p> <ul style="list-style-type: none"> • Encrypt using keys managed through KMS. • Encryption and decryption are handled seamlessly. Users does not have to modify the application to access the data. • Enable encryption while deploying a new cluster via the AWS Management Console or API operations. For additional details see the links below.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/innodb-multiple-tablespaces.html>
- [Encrypting Amazon RDS Resources](#)

Oracle User-Defined Types vs. MySQL User-Defined Types

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		User Defined Types	Aurora MySQL does not support User-Defined Types

Oracle Usage

Oracle refers to User-Defined Types (UDTs) as OBJECT TYPES and are managed using PL/SQL. User Defined Types enable the creation of application-dedicated, complex data types that are based on, and extend, the built-in Oracle data types.

The CREATE TYPE statement supports creating:

- Objects Types
- Varying Array (varray) types
- Nested Table types
- Incomplete Types
- Additional types such as an SQLJ object type (a Java class mapped to SQL user defined type)

Examples

Create an Oracle Object Type to store an employee phone number.

```
CREATE OR REPLACE TYPE EMP_PHONE_NUM AS OBJECT (
    PHONE_NUM VARCHAR2(11));

CREATE TABLE EMPLOYEES (
    EMP_ID    NUMBER PRIMARY KEY,
    EMP_PHONE EMP_PHONE_NUM NOT NULL);

INSERT INTO EMPLOYEES VALUES(1, EMP_PHONE_NUM('111-222-333'));
SELECT a.EMP_ID, a.EMP_PHONE.PHONE_NUM FROM EMPLOYEES a;
```

EMP_ID	EMP_PHONE.P
1	111-222-333

Create an Oracle Object Type as a “collection of attributes” for the employees table.

```
CREATE OR REPLACE TYPE EMP_ADDRESS AS OBJECT (
    STATE    VARCHAR2(2),
    CITY     VARCHAR2(20),
    STREET   VARCHAR2(20),
    ZIP_CODE NUMBER);

CREATE TABLE EMPLOYEES (
```

```

EMP_ID      NUMBER PRIMARY KEY,
EMP_NAME    VARCHAR2(10) NOT NULL,
EMP_ADDRESS EMP_ADDRESS NOT NULL);

INSERT INTO EMPLOYEES VALUES(1, 'John Smith',
EMP_ADDRESS('AL', 'Gulf Shores', '3033 Joyce Street', '36542'));

SELECT a.EMP_ID, a.EMP_NAME, a.EMP_ADDRESS.STATE,
a.EMP_ADDRESS.CITY, a.EMP_ADDRESS.STREET, a.EMP_ADDRESS.ZIP_CODE
FROM EMPLOYEES a;

```

EMP_ID	EMP_NAME	STATE	CITY	STREET	ZIP_CODE
1	John Smith	AL	Gulf Shores	3033 Joyce Street	36542



For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TYPE.html#GUID-E72E3EE6-DE95-4F58-8941-E2F76D0EAE80>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TYPE-BODY.html#GUID-C4F1591A-6F62-4897-9039-2C3F066F1E9D>

MySQL Usage

Currently, Amazon Aurora MySQL does not provide a directly comparable alternative for User-Defined Types.

Oracle Unused Columns vs. MySQL Unused Columns

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL does not support unused columns

Oracle Usage

Oracle provides a method to mark columns as UNUSED. Unused columns are not physically dropped, but are treated as if they were dropped. Unused columns cannot be restored. Select statements do not retrieve data from columns marked as unused and are not displayed when executing a DESCRIBE table command.

The main advantage of setting a column to UNUSED is to reduce possible high database load when dropping a column from a large table. To overcome this issue, a column can be marked as unused and then be physically dropped later.

To set a column as unused, use the SET UNUSED clause.

Examples

```
ALTER TABLE EMPLOYEES SET UNUSED (COMMISSION_PCT);
ALTER TABLE EMPLOYEES SET UNUSED (JOB_ID, COMMISSION_PCT);
```

Display unused columns.

```
SELECT * FROM USER_UNUSED_COL_TABS;
```

```
TABLE_NAME                                COUNT
-----
EMPLOYEES                                3
```

Drop the Column Permanently (physically drop the column).

```
ALTER TABLE EMPLOYEES DROP UNUSED COLUMNS;
```



For more details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.html#GUID-F9CE0CC3-13AE-4744-A43C-EAC7A71AAAB6>

MySQL Usage

Currently, Aurora MySQL does not provide a comparable alternative for unused columns.

Oracle Virtual Columns vs. MySQL Generated Columns

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Create table	Different paradigm and syntax

Oracle Usage

Oracle virtual columns appear as normal columns, but their values are calculated instead of being stored in the database. Virtual columns cannot be created based on other virtual columns and can only reference columns from the same table. When creating a virtual column, you can either explicitly specify the datatype or let the database select the datatype based on the expression.

Notes

- Virtual columns can be used with constraints, indexes, table partitioning, and foreign keys.
- Functions in expressions must be deterministic at the time of table creation.
- Virtual columns cannot be manipulated by DML operations.
- Virtual columns can be used in a WHERE clause and as part of DML commands.
- When creating an index on a virtual column, a function based index is created.
- Virtual columns do not support index-organized tables, external, objects, clusters, or temporary Tables.
- The output of a virtual column expression must be a scalar value.
- The virtual column keyword GENERATED ALWAYS AS and VIRTUAL are not mandatory and are provided for clarity only.

```
COLUMN_NAME [datatype] [GENERATED ALWAYS] AS (expression) [VIRTUAL]
```

- The keyword AS after the column name indicates the column is created as a Virtual Column.
- A virtual column does not need to be specified in an INSERT statement.

Examples

Create a table that includes two virtual columns.

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID NUMBER,
  FIRST_NAME  VARCHAR2(20),
  LAST_NAME   VARCHAR2(25),
  USER_NAME   VARCHAR2(25),
  EMAIL       AS (LOWER(USER_NAME) || '@aws.com'),
  HIRE_DATE   DATE,
  BASE_SALARY NUMBER,
  SALES_COUNT NUMBER,
  FINAL_SALARY NUMBER GENERATED ALWAYS AS
```

```
(CASE WHEN SALES_COUNT >= 10 THEN BASE_SALARY + (BASE_SALARY *
(SALES_COUNT * 0.05)) END) VIRTUAL);
```

Insert a new record into the table without specifying values for the virtual column.

```
INSERT INTO EMPLOYEES
  (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, USER_NAME,
   HIRE_DATE, BASE_SALARY, SALES_COUNT)
VALUES (1, 'John', 'Smith', 'jsmith', '17-JUN-2003', 5000, 21);
```

Select the email virtual column from the table.

```
SELECT email FROM EMPLOYEES;
EMAIL                FINAL_SALARY
-----
jsmith@aws.com      10250
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-TABLE.html#GUID-F9CE0CC3-13AE-4744-A43C-EAC7A71AAAB6>

MySQL Usage

The syntax and functionality of generated columns are similar to virtual columns. They appear as normal columns, but their values are calculated. Generated columns cannot be created based on other Generated Columns and can only reference columns from the same table. When creating a Generated Columns, you must explicitly specify the data type of the column.

Notes

- Unlike Oracle, generated columns can be created based on other generated columns preceding them in the field list.
- Generated columns can be used with constraints, indexes, table partitioning.
- Functions in expressions must be deterministic at the time of table creation.
- Generated columns cannot be manipulated by DML operations.
- Generated columns can be used in a WHERE clause and as part of DML commands.
- When creating an index on a generated column, the generated values are stored in the index.
- The output of a generated column expression must be a scalar value.

Examples

Create a table that includes two generated columns.

```
CREATE TABLE EMPLOYEES (
  EMPLOYEE_ID INT,
  FIRST_NAME  VARCHAR(20),
  LAST_NAME   VARCHAR(25),
  USER_NAME   VARCHAR(25),
  EMAIL       VARCHAR(25) AS
              (CONCAT(LOWER(USER_  NAME), '@aws.com')),
```

```

HIRE_DATE    DATE,
BASE_SALARY  INT,
SALES_COUNT  INT,
FINAL_SALARY INT GENERATED ALWAYS AS
(CASE WHEN SALES_COUNT >= 10 THEN BASE_SALARY + (
  BASE_SALARY * (SALES_COUNT * 0.05)) END) VIRTUAL);

```

Insert a new record into the table without specifying values for the generated column.

```

INSERT INTO EMPLOYEES
  (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, USER_NAME, HIRE_DATE,
   BASE_SALARY, SALES_COUNT)
VALUES(1, 'John', 'Smith', 'jsmith', now(), 5000, 21);

```

Select the email and generated column from the table.

```
SELECT EMAIL, FINAL_SALARY FROM EMPLOYEES;
```

```

email          |FINAL_SALARY |
-----|-----|
jsmith@aws.com |10250        |

```

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-table-generated-columns.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-table-secondary-indexes.html>

Overall Indexes Summary

Usage

MySQL supports multiple types of indexes using different indexing algorithms that can provide performance benefits for different types of queries. The built-in MySQL Index types include:

- **B-Tree**
Default indexes that can be used for equality and range for the majority of queries. These indexes can operate against all datatypes and can be used to retrieve NULL values. B-Tree index values are sorted in ascending order by default.
- **Hash**
Hash Indexes are practical for equality operators. These types of indexes are rarely used because they are not transaction-safe, this type of index is supported by MEMORY and NDB storage engines.
- **Full-Text**
Full-Text indexes are useful when the application needs to query large amount of text, using more complicated morphology attributes.
- **Spatial**
This index supports objects like POINT and GEOMETRY to run geographic related queries

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports descending indexes: DESC in an index definition is no longer ignored but causes storage of key values in descending order. Previously indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others. See [Section 8.3.13 Descending Indexes](#).

CREATE INDEX Synopsis

```

CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part:
    col_name [(length)] [ASC | DESC]

index_option:
    KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_name
    | COMMENT 'string'

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}

```

By default, the CREATE INDEX statement creates a B-Tree index.

Examples

Oracle CREATE/DROP Index:

```

CREATE UNIQUE INDEX IDX_EMP_ID ON EMPLOYEES (EMPLOYEE_ID DESC);
DROP INDEX IDX_EMP_ID;

```

MySQL CREATE/DROP Index:

```

CREATE UNIQUE INDEX IDX_EMP_ID ON EMPLOYEES (EMPLOYEE_ID DESC);
DROP INDEX IDX_EMP_ID;

```

Oracle ALTER INDEX - RENAME:

```

ALTER INDEX IDX_EMP_ID RENAME TO IDX_EMP_ID_OLD;

```

MySQL ALTER INDEX - RENAME:

```

ALTER TABLE EMPLOYEES RENAME INDEX IDX_EMP_ID TO IDX_EMP_ID_OLD;

```

Oracle REBUILD INDEX:

```

ALTER INDEX IDX_EMP_ID REBUILD;

```

MySQL REINDEX (REBUILD) INDEX:

```
ANALYZE TABLE EMPLOYEES;
```



For additional details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-index.htm>
- <https://dev.mysql.com/doc/refman/5.7/en/analyze-table.html>
- <https://dev.mysql.com/doc/refman/5.7/en/alter-table.html>

Summary

Oracle Indexes Types /Features	MySQL Compatibility	MySQL Equivalent
B-Tree Index	Supported	B-Tree Index
Index-Organized Tables	Supported	default behavior by InnoDB
Reverse key indexes	Not supported	-
Descending indexes	Supported	ASC (default) / DESC
B-tree cluster indexes	Not supported	-
Unique / non-unique Indexes	Supported	Syntax is identical
Function-Based Indexes	Not Supported	Use generated columns instead
Application Domain indexes	Not supported	-
BITMAP Index / Bitmap Join Indexes	Not supported	-
Composite Indexes	Supported	Multicolumn Indexes
Invisible Indexes	Not supported	-
Local and Global Indexes	Not supported	-
Partial Indexes for Partitioned Tables (Oracle 12c)	Limited compatibility	Column Prefix index
CREATE INDEX... / DROP INDEX...	Supported	High percentage of syntax similarity
ALTER INDEX... (General Definitions)	Not Supported	-
ALTER INDEX... REBUILD	Supported	ANALYZE TABLE
ALTER INDEX... REBUILD ONLINE	Not Supported	-
Index Metadata	STATISTICS (Oracle USER_INDEXES)	SELECT DISTINCT TABLE_SCHEMA, TABLE_NAME, INDEX_NAME, INDEX_TYPE FROM INFORMATION_SCHEMA.STATISTICS;
Index Tablespace Allocation	Not supported	-
Index Parallel Operations	Not supported	-
Index Compression	No direct equivalent to Oracle index key compression or advanced index compression	-

Oracle Bitmap Indexes vs. MySQL Bitmap Indexes

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	MySQL does not support BITMAP index

Oracle Usage

Bitmap indexes are task-specific indexes best suited for providing fast data retrieval for OLAP workloads and are generally fast for read-mostly scenarios. However, bitmap indexes do not perform well in heavy DML or OLTP workloads.

Unlike B-tree indexes, where an index entry points to a specific table row, a bitmap index stores a bitmap for each index key.

Bitmap indexes are ideal for low-cardinality data filtering where the number of distinct values in a column is relatively small.

Examples

Create an Oracle bitmap index.



```
CREATE BITMAP INDEX IDX_BITMAP_EMP_GEN ON EMPLOYEES (GENDER) ;
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-INDEX.html#GUID-1F89BBC0-825F-4215-AF71-7588E31D8BFE>

MySQL Usage

Currently, Amazon Aurora MySQL does not provide a comparable alternative for bitmap indexes.

Oracle B-Tree Indexes vs. MySQL B-Tree Indexes

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	

Oracle Usage

B-Tree indexes (B stands for balanced), are the most common index type in a relational database and are used for a variety of common query performance enhancing tasks. B-Tree indexes can be defined as an ordered list of values divided into ranges. They provide superior performance by associating a key with a row or range of rows.

B-Tree indexes contain two types of blocks: branch blocks for searching and leaf blocks for storing values. The branch blocks also contain the root branch, which points to lower-level index blocks in the B-Tree index structure.

B-Tree indexes are useful for primary keys and other high-cardinality columns. They provide excellent data access performance for a variety of query patterns such as exact match searches and range searches. B-Tree indexes are the default when creating a new index.

Examples

Create a B-Tree index.

```
CREATE INDEX IDX_EVENT_ID ON SYSTEM_LOG (EVENT_ID);
```

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-INDEX.htm-/#GUID-1F89BBC0-825F-4215-AF71-7588E31D8BFE>

MySQL Usage

MySQL provides full support for B-Tree indexes. Certain constraints created in MySQL (such as primary keys or unique keys) are stored in a B-Tree index format. Similar to Oracle, B-Tree indexes are the default for new indexes.

The query optimizer in MySQL can leverage B-Tree indexes when handling equality and range queries on data. The MySQL optimizer considers using B-Tree indexes to access data, especially when queries use one or more of the following operators: >, >=, <, <=, =.

In addition, query elements such as IN, BETWEEN, IS NULL or IS NOT NULL can also leverage B-Tree indexes for faster data retrieval.

There are two types of indexes:

- **Clustered Index:** A reference as primary key. When a primary key is defined on a table, InnoDB uses it as the clustered index. It is highly recommended to specify a primary key for all tables. If there is no primary key, MySQL locates the first UNIQUE index where all columns are NOT NULL and are used as a clustered index. If there is no primary key or UNIQUE index to use, InnoDB internally generates a hidden clustered index named GEN_CLUST_INDEX.

- **Secondary Index:** All indexes that are not clustered indexes. Each index entry has a reference to the clustered index. If the clustered index is applied on long values, the secondary indexes consume more storage space.



Examples

Create a B-Tree index.

```
CREATE INDEX IDX_EVENT_ID ON SYSTEM_LOG (EVENT_ID);  
OR  
CREATE INDEX IDX_EVENT_ID ON SYSTEM_LOG (EVENT_ID) USING BTREE;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/create-index.htm>

Oracle Composite Indexes vs. MySQL Multiple-Column Indexes

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	

Oracle Usage

An index created on multiple table columns is known as a multi-column, concatenated, or composite index. The main purpose of composite indexes is to improve the performance of data retrieval for SELECT statements when filtering on all, or some, of the composite index columns. When using composite indexes, it is beneficial to place the most restrictive columns at the first position of the index to improve query performance. Column placement order is crucial when using composite indexes because the most prevalent columns are accessed first.

Examples

Create a composite index on the HR.EMPLOYEES table.

```
CREATE INDEX IDX_EMP_COMPI ON
      EMPLOYEES (FIRST_NAME, EMAIL, PHONE_NUMBER);
```

Drop a composite index.

```
DROP INDEX IDX_EMP_COMPI;
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/indexes-and-index-organized-tables.html#GUID-ABE1DE2A-59CC-4ADE-86A5-426B16459464>

MySQL Usage

MySQL multiple-column indexes are similar to composite indexes in Oracle.

These indexes are beneficial when queries filter on all indexed columns, the first indexed column, the first two indexed columns, the first three indexed columns, and so on. When indexed columns are specified in the optimal order during index creation, a single multiple-column index can improve performance in scenarios where several queries access the same database table.

Up to 16 columns can be specified when creating a multiple-column index.

Examples

Create a multiple-column index on the EMPLOYEES table.



```
CREATE INDEX IDX_EMP_COMPI ON EMPLOYEES
      (FIRST_NAME, EMAIL, PHONE_NUMBER);
```

Drop a multiple-column Index.

```
DROP INDEX IDX_EMP_COMPI ON EMPLOYEES;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/multiple-column-indexes.html>

Oracle Function-Based Indexes vs. MySQL Indexing on Generated Columns

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	MySQL doesn't support functional indexes; a workaround is available

Oracle Usage

Function-based indexes allow functions to be used in the WHERE clause of queries on indexed columns. Function-based indexes store the output of a function applied on the values of a table column. The Oracle query optimizer only uses a function-based index when the function is used as part of a query.

Oracle evaluates the return values of functions when it processes DML in order to keep the index synchronized with the table values.

Examples

Create a function-based index.

```
CREATE TABLE SYSTEM_EVENTS (
    EVENT_ID NUMERIC PRIMARY KEY,
    EVENT_CODE VARCHAR2(10) NOT NULL,
    EVENT_DESCRIPTION VARCHAR2(200),
    EVENT_TIME TIMESTAMP NOT NULL);

CREATE INDEX EVNT_BY_DAY ON SYSTEM_EVENTS (
    EXTRACT(DAY FROM EVENT_TIME));
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/indexes-and-index-organized-tables.html#GUID-797E49E6-2DCE-4FD4-8E4A-6E761F1383D1>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-INDEX.html#GUID-1F89BBC0-825F-4215-AF71-7588E31D8BFE>

MySQL Usage

MySQL does not directly support a feature equivalent to Oracle function-based indexes. However, workarounds exist that can offer similar functionality. Specifically, you can create secondary indexes on MySQL generated columns. Implementing this workaround may require modification of existing SQL queries.

A generated column derives its values from the result of an expression. Creating an index on a generated column allows the generated column to be used in a WHERE clause of a query while accessing data with the index. Unlike Oracle function-based indexes, this workaround requires specifying the function in the table column specification.

Notes

- Generated columns can be created as either STORED or VIRTUAL. For our purposes, we need to create generated columns as STORED. Otherwise, we won't be able to index those columns.
- MySQL can't use stored routines or functions with generated columns.
- Generated columns support NOT NULL restrictions.
- A generated expression cannot exceed 64K for the entire table. For example, you can create a single field with a generated expression length of 64K or 12 fields with a length of 5K each.
- A generated column cannot refer to itself or to other generated columns defined later, but it can refer to any previously defined generated columns.
- The generation expression can only call native deterministic functions.
- You can mix VIRTUAL and STORED columns within a table.
- When inserting data to the table, you should not reference the generated columns in your insert statement.

Examples

First example:

Create a generated column that calculates the yearly salary based on the monthly salary, and create a secondary index on that column.

```
CREATE TABLE EMPS (ID INT, MONTH_SALARY INT,
  YEAR_SALARY INT GENERATED ALWAYS AS (MONTH_SALARY*12),
  INDEX FBI_YEAR_IDX (YEAR_SALARY));
```

```
INSERT INTO EMPS (ID, MONTH_SALARY) VALUES (1,10000);
INSERT INTO EMPS (ID, MONTH_SALARY) VALUES (2,8764);
INSERT INTO EMPS (ID, MONTH_SALARY) VALUES (3,4355);
INSERT INTO EMPS (ID, MONTH_SALARY) VALUES (4,6554);
```

```
SELECT * FROM EMPS;
```

ID	MONTH_SALARY	YEAR_SALARY
1	10000	120000
2	8764	105168

Queries can reference the YEAR_SALARY column as part of the WHERE clause and access data using the FBI_YEAR_IDX index.

```
SELECT * FROM EMPS WHERE YEAR_SALARY>80000;
```

```
SELECT * FROM EMPS WHERE MONTH_SALARY*12>80000;
```

Second example:

Create two generated columns using string manipulation functions as part of the table specification with secondary indexes on each.

```
CREATE TABLE EMPS (ID INT, FULL_NAME CHAR(40),
  FIRST_NAME CHAR(20) GENERATED ALWAYS AS
  (SUBSTRING(FULL_NAME, 1, INSTR(FULL_NAME, ' '))),
  LAST_NAME CHAR(20) GENERATED ALWAYS AS
  (SUBSTRING(FULL_NAME, INSTR(FULL_NAME, ' '))),
  INDEX FBI_FNAME_IDX (FIRST_NAME),
  INDEX FBI_LNAME_IDX (LAST_NAME));
```

```
INSERT INTO EMPS (ID, FULL_NAME) VALUES (1,'James Kirk');
INSERT INTO EMPS (ID, FULL_NAME) VALUES (2,'Benjamin Sisko');
INSERT INTO EMPS (ID, FULL_NAME) VALUES (3,'Karthryn Janeway');
INSERT INTO EMPS (ID, FULL_NAME) VALUES (4,'Jean- Luc Picard');
```

Queries can now use the FBI_FNAME_IDX index :

```
SELECT ID FROM EMPS WHERE
  SUBSTRING(FULL_NAME, 1, INSTR(FULL_NAME, ' '))='Jacob';
```



```
SELECT ID FROM EMPS WHERE FIRST_NAME='Jacob';
```

Note: For this specific example, generated columns were not necessary but provided as an example. Instead, a B-tree index created on the column prefix could have been used to achieve the same results.

```
CREATE TABLE EMPS (ID INT, FULL_NAME CHAR(40));  
CREATE INDEX FBI_NAME_PREF_IDX ON EMPS (FULL_NAME (20));
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/create-table-generated-columns.html>

Oracle Invisible Indexes vs. MySQL Invisible Indexes

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	MySQL does not support invisible index

Oracle Usage

The Oracle invisible index feature gives database administrators the ability to create indexes, or change existing indexes, that are ignored by the optimizer. They are maintained during DML operations and are kept relevant, but are different from usable indexes.

The most common uses for invisible indexes are:

- Testing the effect of a dropped index without actually dropping it.
- Using a specific index for certain operations or modules of an application without affecting the overall application.
- Adding an index to a set of columns on which an index already exists.

Notes:

- Database administrators can force the optimizer to use invisible indexes by changing the `OPTIMIZER_USE_INVISIBLE_INDEXES` parameter to true.
- Invisible indexes can be used if specified as a HINT.

Examples

Change an index to an invisible index.

```
ALTER INDEX idx_name INVISIBLE;
```

Change an invisible index to a visible index.

```
ALTER INDEX idx_name VISIBLE;
```

Create an invisible index.

```
CREATE INDEX idx_name ON employees(first_name) INVISIBLE;
```

Query all invisible indexes.



```
SELECT TABLE_OWNER, INDEX_NAME FROM DBA_INDEXES
WHERE VISIBILITY = 'INVISIBLE';
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-indexes.html#GUID-3A66938F-73C6-4173-844E-3938A0DBBB54>

MySQL Usage

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports invisible indexes. An invisible index is not used by the optimizer at all but is otherwise maintained normally. Indexes are visible by default. Invisible indexes make it possible to test the effect of removing an index on query performance without making a destructive change that must be undone should the index turn out to be required. See [Section 8.3.12 Invisible Indexes](#).

Oracle Index-Organized Table (IOT) vs. MySQL InnoDB Clustered Index

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	MySQL does not support the index-organized tables object due to this is the default behavior for InnoDB

Oracle Usage

The Oracle index-organized table (IOT) object is a special type of index/table hybrid that physically controls how data is stored at the table and index level. When creating a common database table, or heap-organized table, the data is stored unsorted (as a heap). However, when creating an index-organized table, the actual table data is stored in a B-tree index structure sorted by the primary key of each row. Each leaf block in the index structure stores both the primary key and non-key columns.

IOTs provide performance improvements when accessing data using the primary key because table records are sorted (clustered) using the primary key and physically co-located alongside the primary key.

Examples

Create an Oracle index-organized table storing ordered data based on the primary key.

```
CREATE TABLE SYSTEM_EVENTS (
  EVENT_ID NUMBER,
  EVENT_CODE VARCHAR2(10) NOT NULL,
  EVENT_DESCRIPTION VARCHAR2(200),
  EVENT_TIME DATE NOT NULL,
  CONSTRAINT PK_EVENT_ID PRIMARY KEY(EVENT_ID))
  ORGANIZATION INDEX;

INSERT INTO SYSTEM_EVENTS VALUES (9, 'EVNT-A1-10', 'Critical', '01-JAN-2017');
INSERT INTO SYSTEM_EVENTS VALUES (1, 'EVNT-C1-09', 'Warning', '01-JAN-2017');
INSERT INTO SYSTEM_EVENTS VALUES (7, 'EVNT-E1-14', 'Critical', '01-JAN-2017');
SELECT * FROM SYSTEM_EVENTS;
```

EVENT_ID	EVENT_CODE	EVENT_DESCRIPTION	EVENT_TIM
1	EVNT-C1-09	Warning	01-JAN-17
7	EVNT-E1-14	Critical	01-JAN-17
9	EVNT-A1-10	Critical	01-JAN-17

Note: The records are sorted in the reverse order from which they were inserted.

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/indexes-and-index-organized-tables.html#GUID-797E49E6-2DCE-4FD4-8E4A-6E761F1383D1>

MySQL Usage

MySQL does not support index-organized tables. However it provides similar functionality using InnoDB (the Amazon Aurora default storage engine).

Each InnoDB table provides a special clustered index. When you create a PRIMARY KEY on a table, InnoDB automatically uses it as the clustered index. This behavior is similar to index-organized tables in Oracle.

It is best practice to specify a primary key for each MySQL table. If you do not specify a primary key, MySQL locates the first unique index where all key columns are specified as NOT NULL and uses it as the clustered index.

If a table layout does not logically provide a column (or multiple columns) that is both unique and not null, it is recommended to explicitly add an auto-incremented column to generate unique values.

Note: If no primary key or a suitable unique index can be found, InnoDB actually creates a hidden GEN_CLUST_INDEX clustered index with internally generated row ID values. These auto-generated row IDs are based on a 6-byte field that increases monotonically.

Examples

Create a new table with a simple primary key. Because the storage engine is InnoDB, the table is created as a clustered table sorting data based on the primary key itself.

```
CREATE TABLE SYSTEM_EVENTS (
  EVENT_ID INT PRIMARY KEY,
  EVENT_CODE VARCHAR(10) NOT NULL,
  EVENT_DESCRIPTION VARCHAR(200),
  EVENT_TIME DATE NOT NULL);



INSERT INTO SYSTEM_EVENTS VALUES (9, 'EVNT10', 'Critical', NOW());
INSERT INTO SYSTEM_EVENTS VALUES (1, 'EVNT09', 'Warning', NOW());
INSERT INTO SYSTEM_EVENTS VALUES (7, 'EVNT14', 'Critical', NOW());

SELECT * FROM SYSTEM_EVENTS;
```

EVENT_ID	EVENT_CODE	EVENT_DESCRIPTION	EVENT_TIME
1	EVNT-C1-09	Warning	2018-02-14
7	EVNT-E1-14	Critical	2018-02-14
9	EVNT-A1-10	Critical	2018-02-14

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/innodb-index-types.html>

Oracle Local and Global Partitioned Indexes vs. MySQL Partitioned Indexes

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Indexes	MySQL does not support global indexes

Oracle Usage

Local and global indexes are used for partitioned tables in Oracle databases. Each index created on a partitioned table can be specified as either local or global.

- **Local Partitioned Index:** Maintains a one-to-one relationship between the index partitions and the table partitions. For each table partition, a separate index partition is created. This type of index is created using the LOCAL clause. Because each index partition is independent, index maintenance operations are easier and can be performed independently. Local partitioned indexes are managed automatically by Oracle during creation or deletion of table partitions.
- **Global Partitioned Index:** Each global index contains keys from multiple table partitions in a single index partition. This type of index is created using the GLOBAL clause during index creation. A global index can be partitioned or non-partitioned (default). Certain restrictions exist when creating global partitioned indexes on partitioned tables, specifically for index management and maintenance. For example, dropping a table partition causes the global index to become unusable without an index rebuild.

Examples

Create a local and global index on a partitioned table.

```
CREATE INDEX IDX_SYS_LOGS_LOC ON SYSTEM_LOGS (EVENT_DATE)
LOCAL
(PARTITION EVENT_DATE_1,
PARTITION EVENT_DATE_2,
PARTITION EVENT_DATE_3);
```

```
CREATE INDEX IDX_SYS_LOGS_GLOB ON SYSTEM_LOGS (EVENT_DATE)
GLOBAL PARTITION BY RANGE (EVENT_DATE) (
PARTITION EVENT_DATE_1 VALUES LESS THAN (TO_DATE
('01/01/2015', 'DD/MM/YYYY')),
PARTITION EVENT_DATE_2 VALUES LESS THAN (TO_DATE('01/01/2016', 'DD/MM/YYYY')),
PARTITION EVENT_DATE_3 VALUES LESS THAN (TO_DATE('01/01/2017', 'DD/MM/YYYY')),
PARTITION EVENT_DATE_4 VALUES LESS THAN (MAXVALUE));
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/vldb/partition-concepts.html#GUID-EA7EF5CB-DD49-43AF-889A-F83AAC0D7D51>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/vldb/index-partitioning.html#GUID-569F94D0-E6E5-45BB-9626-5506DE18FF00>

MySQL Usage

Indexes created on a partitioned tables are similar to local indexes in Oracle. MySQL does not provide an equivalent for Oracle global indexes because in MySQL, partitioning applies to all data and indexes of a table. It is not possible to partition only the data and not the indexes. All indexes on partitioned tables behave like an Oracle local index.

Examples

Drop a partition (the index is being used without a rebuild). Note that the execution plan shows the scanned partitions.

```
ALTER TABLE SYSTEM_LOGS add INDEX EVENT_NO_IDX (EVENT_NO);
EXPLAIN SELECT * from SYSTEM_LOGS where EVENT_NO=2;
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered
1	SIMPLE	SYSTEM_LOGS	warning,critical	ref	EVENT_NO_IDX	EVENT_NO_IDX	4	const	1	100

```
ALTER TABLE SYSTEM_LOGS DROP PARTITION critical;
EXPLAIN SELECT * from SYSTEM_LOGS where EVENT_NO=2;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered
1	SIMPLE	SYSTEM_LOGS	warning	ref	EVENT_NO_IDX	EVENT_NO_IDX	4	const	1	100

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html>

Automatic Indexing

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Index	MySQL does provide a Automatic Indexing feature

Oracle Usage

Oracle 19 introduces automatic indexing feature. This feature automates the index management tasks by automatically creating, rebuilding, and dropping indexes based on the changes in application workload, thus improving database performance.

Important functionality provided by Automatic Indexing:

- Automatic indexing process runs in the background at a predefined time interval and analyzes application workload. It identifies the tables/columns that are candidates for new indexes and creates new indexes.
- The auto indexes as initially created as invisible indexes. These invisible auto indexes are verified against SQL statements and if the performance is improved, then these indexes are converted as visible indexes.
- Identify and drop any existing under-performing auto indexes or any auto indexes not used for long period.
- Rebuilds the auto indexes that are marked unusable due to DDL operations.
- Provides package `DBMS_AUTO_INDEX` to configure automatic indexing and for generating reports related to automatic indexing operations.

Note: Up to date table statistics are very important for the Auto indexing to function efficiently. Tables without statistics or with stale statistics are not considered for auto indexing

Package `DBMS_AUTO_INDEX` is used to configuring auto indexes and generating reports.

Below are some of the configuration options which can be set by using `CONFIGURE` procedure of `DBMS_AUTO_INDEX` package:

- Enabling and disabling automatic indexing in a database
- Specifying schemas and tables that can use auto indexes
- Specifying a retention period for unused auto indexes. By default, the unused auto indexes are deleted after 373 days
- Specifying a retention period for unused non-auto indexes
- Specifying a tablespace and a percentage of tablespace to store auto indexes

Below are some of the reports related to automatic indexing operations which can be generated using `REPORT_ACTIVITY` and `REPORT_LAST_ACTIVITY` functions of the `DBMS_AUTO_INDEX` package.

- Report of automatic indexing operations for a specific period
- Report of the last automatic indexing operation

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-indexes.html#GUID-E4149397-FF37-4367-A12F-675433715904>

MySQL Usage

Currently, Amazon Aurora MySQL does not provide a comparable alternative for Automatic Indexing

To most reasonable option would be to run a scheduled set of queries to estimate if additional indexes are needed.

The following queries can help determine that

Find user-tables without primary keys

```
SELECT tab.table_schema,tab.table_name
FROM information_schema.tables tab
LEFT JOIN information_schema.table_constraints tco
      ON tab.table_schema = tco.table_schema
      AND tab.table_name = tco.table_name
      AND tco.constraint_type = 'PRIMARY KEY'
WHERE tco.constraint_type is null
      AND tab.table_schema not in('information_schema', 'performance_schema', 'sys')
      AND tab.table_type = 'BASE TABLE'
ORDER BY tab.table_schema, tab.table_name;
```



Unused indexed that can probably be dropped

```
SELECT * FROM sys.schema_unused_indexes;
```

All of these should not be implemented in a script to decide if to create or drop indexes in a production environment, the Oracle Automatic indexes will first assess if a new index is needed and if so, it will create an invisible index and only after ensuring not was harmed then the index will become visible, similar process can't be used in MySQL to avoid any production performance issues.

Special Features & Future Content

Oracle Advanced Queuing vs. MySQL Integration with Lambda

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Use AWS Lambda and SQS with Aurora MySQL

Oracle Usage

The Oracle Advanced Queuing (AQ) feature enables database-integrated message queuing functionality. It is based on Oracle Streams and optimizes data functions by storing messages, allocating the messages to different service queues, and transmitting the messages using Oracle Net Services, HTTP, and HTTPS. AQ is implemented using database tables.

Oracle provides the Java package `oracle.jdbc.aq` as an interface to AQ. It contains the following items:

- **Classes:**

- AQDequeueOptions - Specifies the options for the dequeue operation.
- AQEnqueueOptions - Specifies the options for the enqueue operation.
- AQFactory - A factory class for AQ, which creates components such as agent or message properties.
- AQNotificationEvent - New message notifications.

- **Interfaces:**

- AQAgent - An identity of a user, producer, or consumer of a message.
- AQMessage - An enqueued or dequeued message.
- AQMessageProperties - Message properties such as:
 - Correlation
 - Sender
 - Delay
 - Expiration
 - Recipients
 - Priority
 - Ordering
- AQNotificationListener - A listener interface for receiving AQ notification events.
- AQNotificationRegistration - A registration to be notified when a new message is enqueued on a particular queue.

For more information and code examples, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/adque/>

MySQL Usage



Aurora MySQL provides built-in integration with Lambda functions, which can be called from within the database and interact with AWS Messaging Services. The integration with Lambda functions provides a powerful framework for leveraging AWS services to implement custom solutions with less code.

Examples

See [RDS Simple Notification Service \(SNS\)](#) for an example of sending an email message from within a database.

For more details, see [Invoking a Lambda Function with an Aurora MySQL Native Function](#)

Oracle Character Sets vs. MySQL Character Sets and Collations

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Different syntax MySQL can have different collation for each database in the same instance

Oracle Usage

Oracle supports most national and international encoded character set standards including extensive support for Unicode.

Oracle provides two scalar string-specific data types:

- **VARCHAR2:** Stores variable-length character strings with a length between 1 and 4000 bytes. An Oracle database can be configured to use the VARCHAR2 data type to store either Unicode or Non-Unicode characters.
- **NVARCHAR2:** Scalar data type used to store Unicode data. Supports AL16UTF16 or UTF8 and is specified during database creation.

Character sets in Oracle are defined at the Instance level (Oracle 11g) or the Pluggable Database level (Oracle 12c R2). In Pre-12cR2 Oracle databases, the character set for the root Container and all Pluggable Databases were required to be identical.

Oracle 18c updates AL32UTF8 and AL16UTF16 character sets to Unicode standard version 9.0

UTF8 Unicode

Oracle's implementation of UTF8 Unicode uses the AL32UTF8 Character Set and provides encoding of ASCII characters as single-byte for Latin characters, two-bytes for some European and Middle-Eastern languages, and three-bytes for certain South and East-Asian characters. Therefore, Unicode storage requirements are usually higher when compared non-Unicode character sets.

Character Set Migration

Two options exist for modifying existing Instance-level or database-level character sets:

- Export/Import from the source Instance/PDB to a new Instance/PDB with a modified character set.
- Use the Database Migration Assistant for Unicode (DMU), which simplifies the migration process to the Unicode character set.

As of 2012, use of the CSALTER utility for character set migrations is deprecated.

Notes:

- Oracle Database 12c Release 1 (12.1.0.1) complies with version 6.1 of the Unicode Standard.
- Oracle Database 12c Release 2 (12.1.0.2) extends the compliance to version 6.2 of the Unicode standard.
- UTF-8 is supported through the AL32UTF8 CS and is valid as both the client and database character sets.
- UTF-16BE is supported through AL16UTF16 and is valid as the national (NCHAR) character set.

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/nlspg/choosing-character-set.htm#GUID-BF26E01D-AB92-48FC-855A-69A5B3AF9A92>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/nlspg/appendix-A-locale-data.htm#GUID-A9E30C27-FD47-4552-B670-F41A95B11405>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/nlspg/supporting-multilingual-databases-with-unicode.html#GUID-AA09A60E-123E-457C-ACE1-89E4634E492C>

MySQL Usage

MySQL supports a variety of different character sets including support for both single-byte and multi-byte languages. The default character set is specified when initializing a MySQL database cluster with initdb. Each individual database created on the MySQL cluster supports individual character sets defined as part of database creation.

To query the available character sets, use the INFORMATION_SCHEMA CHARACTER_SETS table or the SHOW CHARACTER SET statement.

All character sets have at least one collation, and most character sets have more. To list the display collations for a character set, use the INFORMATION_SCHEMA COLLATIONS table or the SHOW COLLATION statement.

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has a default collation.
- Collation names start with the name of the character set with which they are associated and are generally followed by one or more suffixes indicating other collation characteristics.

Examples

Create a database named test01 that uses the Korean EUC_KR Encoding and the euckr_korean_ci collation.

```
CREATE DATABASE test01 CHARACTER SET = euckr COLLATE = euckr_korean_ci;
```

View the character sets configured for each database by querying the System Catalog.

```
SELECT SCHEMA_NAME, DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME FROM INFORMATION_SCHEMA.SCHEMATA;
```

Convert a character set and collation using the ALTER DATABASE command.

```
ALTER DATABASE test01 CHARACTER SET = ucs2 COLLATE = ucs2_general_ci;
```

MySQL supports conversion of character sets between server and client for specific character set combinations with the parameter `character_set_client` and `character_set_connection`. For more information see <https://dev.mysql.com/doc/refman/5.7/en/charset-connection.html>.

MySQL supports specifying the sort order and character classification behavior on a per-column level. Specify specific collations for individual table columns.

```
CREATE TABLE lang(
latin1_col CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci,
latin2_col CHAR(10) CHARACTER SET latin2);
```


Summary

	Oracle	Aurora MySQL
View database character set	SELECT * FROM NLS_DATABASE_PARAMETERS;	SELECT SCHEMA_NAME, DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME FROM INFORMATION_SCHEMA.SCHEMATA;
Modify the database character set	1. Full Export/Import. 2. When converting to Unicode, use the Oracle DMU utility.	ALTER DATABASE test01 CHARACTER SET = ucs2 COLLATE = ucs2_general_ci;
Character set granularity	Instance (11g + 12cR1) Database (Oracle 12cR2)	column
UTF8	Supported via VARCHAR2 and NVARCHAR data types	Supported via CHAR and VARCHAR
UTF16	Supported via NVARCHAR2 datatype	Supported via CHAR and VARCHAR
NCHAR/NVARCHAR data types	Supported	Supported

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/charset.html>
- <https://dev.mysql.com/doc/refman/5.7/en/charset-database.html>

Oracle Database Links vs. MySQL Fully-Qualified Table Names

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	MySQL does not support database links

Oracle Usage

Database Links are schema objects used to interact with remote database objects. Common use cases for database links include selecting data from tables that reside in remote databases.

Note: To use database links, Oracle net services must be installed on both the local and remote database servers to facilitate communications.

Examples

Create a database link named `remote_db`. When creating a database link, you have the option of specifying the remote database destination as a TNS Entry or a full TNS Connection string.

```
CREATE DATABASE LINK remote_db CONNECT TO username IDENTIFIED BY password USING
'remote';

CREATE DATABASE LINK remotenoTNS CONNECT TO username IDENTIFIED BY password USING '
(DESCRIPTION=(ADDRESS_LIST=(ADDRESS = (PROTOCOL = TCP)(HOST =192.168.1.1)(PORT =
1521))) (CONNECT_DATA =(SERVICE_NAME = orcl)))';
```

After the database link is created, you can use the database link directly as part of a SQL query using the database link name (`@remote_db`) as a suffix to the table name.

```
SELECT * FROM employees@remote_db;
```

Database links also support DML commands.

```
INSERT INTO employees@remote_db
(employee_id, last_name, email, hire_date, job_id) VALUES
(999, 'Claus', 'sclaus@example.com', SYSDATE, 'SH_CLERK');

UPDATE jobs@remote_db SET min_salary = 3000 WHERE job_id = 'SH_CLERK';

DELETE FROM employees@remote_db WHERE employee_id = 999;
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-a-distributed-database.html#GUID-7B0C4627-4473-4313-88D5-FD03CA42D9EA>

MySQL Usage

Currently, MySQL does not provide a direct comparable alternative for Oracle Database Links. You can use the fully-qualified names to query data from another database within the same cluster. This functionality is similar to Oracle's option that allows you to query data from a different schema. If the data cannot be stored under the same MySQL Cluster, then there is no equivalent to Oracle's Database Links in MySQL.

If the data cannot be placed under the same MySQL Cluster then there is no relevant equivalent to Oracle's Database Links in MySQL.


Examples

Query all flight ids from the all_flights table in the flights database, assume that this is being executed from another database.

```
SELECT flight_id from flights.all_flights;
```

Note: This query returns the data only if the user has permissions to the table and the database.

Oracle DBMS_SCHEDULER vs. MySQL Events

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Different paradigm and syntax

Oracle Usage

The DBMS_SCHEDULER package contains a collection of scheduling functions that can be executed or called from PL/SQL.

There are two main objects involved with creating scheduling jobs: PROGRAM and SCHEDULE. A program defines what to execute, and a schedule defines when to execute the program. The Scheduler can run a database program unit (such as a procedure) or an external executable (such as files system shell scripts).

There are three execution methods associated with jobs: Time-Based Scheduling, Event-Based Scheduling, and Dependency Jobs. Examples of each are provided in the following section.

Examples

Time Base Scheduling

Create a job with a program and a schedule.

1. Create a program that calls the procedure UPDATE_HR_SCHEMA_STATS in the HR schema.
2. Create a schedule that sets the Interval. This example runs the job every hour.
3. Create the job.

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM(
program_name => 'CALC_STATS',
program_action => 'HR.UPDATE_HR_SCHEMA_STATS',
program_type => 'STORED_PROCEDURE',
enabled => TRUE);
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE(
schedule_name => 'stats_schedule',
start_date => SYSTIMESTAMP,
repeat_interval => 'FREQ=HOURLY;INTERVAL=1',
comments => 'Every hour');
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name          => 'my_new_job3',
program_name      => 'my_saved_program1',
```

```

schedule_name      => 'my_saved_schedule1');
END;
/

```

Create a job without a program or schedule.

1. **job_type**: EXECUTABLE – The job runs a an external script.
2. **job_action**: Specify the location of the external script.
3. **start_date**: Specify when the job will be enabled.
4. **repeat_interval**: Specify when the job will run. In this example, it runs every day at hour 23 (11:00 PM).

```

BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name=>'HR. BACKUP',
job_type => 'EXECUTABLE',
job_action => '/home/usr/dba/rman/nightly_bck.sh',
start_date=> SYSDATE,
repeat_interval=>'FREQ=DAILY;BYHOUR=23',
comments => 'Nightly backups');
END;
/

```

After the job is created, update its attributes with the SET_ATTRIBUTE procedure.

```

BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
name          => 'my_emp_job1',
attribute     => 'repeat_interval',
value        => 'FREQ=DAILY');
END;
/

```

Event-Based Scheduling

Create a schedule to start a job whenever the Scheduler receives an event indicating a file arrived on the system before 9 AM, and then create a job to use the schedule.

```

BEGIN
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
schedule_name      => 'scott.file_arrival',
start_date         => systimestamp,
event_condition    => 'tab.user_data.object_owner = ''SCOTT''
and tab.user_data.event_name = ''FILE_ARRIVAL''
and extract hour from tab.user_data.event_timestamp < 9',
queue_spec         => 'my_events_q');
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name           => my_job,
program_name       => my_program,
start_date         => '15-JUL-04 1.00.00AM US/Pacific',
event_condition    => 'tab.user_data.event_name = ''LOW_INVENTORY''',

```

```

queue_spec          => 'my_events_q'
enabled             => TRUE,
comments           => 'my event-based job');
END;
/

```

Dependency Jobs (Chained)

1. Use DBMS_SCHEDULER.CREATE_CHAIN to create a chain.
2. Use DBMS_SCHEDULER.DEFINE_CHAIN_STEP to define three steps for the chain. The referenced programs must be enabled.
3. Use DBMS_SCHEDULER.DEFINE_CHAIN_RULE to define corresponding rules for the chain.
4. Use DBMS_SCHEDULER.ENABLE to enable the chain.
5. Use DBMS_SCHEDULER.CREATE_JOB to create a chain job that starts daily at 1:00 PM.

```

BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
chain_name          => 'my_chain1',
rule_set_name       => NULL,
evaluation_interval => NULL,
comments           => NULL);
END;
/

BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepA', 'my_program1');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepB', 'my_program2');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepC', 'my_program3');
END;
/

BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain1', 'TRUE', 'START stepA');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
'my_chain1', 'stepA COMPLETED', 'Start stepB, stepC');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
'my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END');
END;
/

BEGIN
DBMS_SCHEDULER.ENABLE('my_chain1');
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name           => 'chain_job_1',
job_type           => 'CHAIN',
job_action         => 'my_chain1',
repeat_interval    => 'freq=daily;byhour=13;byminute=0;bysecond=0',
enabled            => TRUE);

```

```
END;
/
```

There are two additional objects associated with jobs:

- **JOB CLASS:** When you have a number of jobs that have the same behavior and attributes, you can group them together into a logical group called a “Job Class”. You can specify priorities between job classes by allocating percentages of available resources.
- **WINDOW:** To prioritize jobs based on schedule, you can create a “window” of time when the jobs can run. For example, non-peak time hours or at end of the month.

For additional details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/scheduling-jobs-with-oracle-scheduler.html#GUID-D41660D0-D88F-4D9F-8CC8-63D040EDC4E6>

MySQL Usage

Aurora MySQL can use EVENT objects to run scheduled events in the database. It can run a one-time event or a repeated event (cycled). A repeated event is a time-base trigger that executes SQL, executes commands, or calls a procedure.

The event_scheduler parameter must be turned ON (not the default value) to use this feature.

If an EVENT terminates with errors, it is written to the error log. If there is a need to simulate the dba_scheduler_job_log, the error log can be defined to use TABLE as the output. For additional information, see [Error Log via Amazon RDS Console](#).

Examples

Check if the event scheduler process is enabled.

```
select @@GLOBAL.event_scheduler
```

View all events.

```
select * from INFORMATION_SCHEMA.EVENTS;
```

Create a new event that executes a procedure every minute.

```
CREATE EVENT event_exec_myproc ON SCHEDULE EVERY 1 MINUTE
DO CALL simpleproc1(5);
```

Summary

Description	Oracle Scheduler	MySQL Events
Create a job that runs s Stored Procedure	BEGIN DBMS_SCHEDULER.CREATE_PROGRAM(program_name => 'CALC_STATS', program_ action => 'HR.UPDATE_HR_SCHEMA_	CREATE EVENT stats_schedule ON SCHEDULE EVERY 1 HOUR DO CALL HR.UPDATE_HR_ SCHEMA_STATS();



Description	Oracle Scheduler	MySQL Events
	<pre> STATS', program_type => 'STORED_ PROCEDURE', enabled => TRUE); END; / BEGIN DBMS_SCHEDULER.CREATE_SCHEDULE(schedule_name => 'stats_schedule', start_date => SYSTIMESTAMP, repeat_interval => 'FREQ- Q=HOURLY;INTERVAL=1', comments => 'Every hour'); END; / BEGIN DBMS_SCHEDULER.CREATE_JOB (job_ name => 'my_new_job3', program_name => 'my_ saved_program1', schedule_name => 'my_ saved_schedule1'); END; / </pre>	
<p>Create a job that run external executables</p>	<pre> BEGIN DBMS_SCHEDULER.CREATE_PROGRAM (program_name => 'oe.my_saved_program1', pro- gram_action => '/usr/local/bin/date', program_ type => 'EXECUTABLE', comments => 'My com- ments here'); END; / </pre>	<p>To run Lambda function:</p> <pre> CALL mysql.lambda_async ('arn:aws:lambda:us-west- 2:123456789012:function:oe.my_ saved_program1',{'input1": "value"}') </pre> <p>For additional information: click</p> <p>Note: The lambda_async function runs a lambda function and gets a JSON object for the input values.</p>
<p>Create event-base job</p>	<pre> BEGIN DBMS_SCHEDULER.CREATE_EVENT_ SCHEDULE (schedule_name => 'scott.file_ arrival', start_date => systimestamp, event_con- dition => 'tab.user_data.object_owner = "SCOTT" and tab.user_data.event_name = "FILE_ARRIVAL" and extract hour from tab.user_ data.event_timestamp < 9', queue_spec => 'my_ events_q'); END; / BEGIN DBMS_SCHEDULER.CREATE_JOB (job_ name => my_job, program_name => my_pro- gram, start_date => '15-JUL-04 1.00.00AM </pre>	<p>For the CREATE EVENT syntax, only time intervals can be defined as triggers for the event.</p> <p>If an event job is required, the best alternatives are:</p> <ol style="list-style-type: none"> 1. Create triggers to execute the commands (for DML events). 2. Create an EVENT that runs every X time and check if the event occurred (the minimum interval is one second).

Description	Oracle Scheduler	MySQL Events
	<pre>US/Pacific', event_condition => 'tab.user_ data.event_name = "LOW_INVENTORY"', queue_spec => 'my_events_q' enabled => TRUE, comments => 'my event-based job'); END; /</pre>	
Create chained-base job	<pre>BEGIN DBMS_SCHEDULER.CREATE_CHAIN (chain_name => 'my_chain1', rule_set_name => NULL, evaluation_interval => NULL, comments => NULL); END; / BEGIN DBMS_SCHEDULER.DEFINE_CHAIN_STEP ('my_chain1', 'stepA', 'my_program1'); DBMS_SCHEDULER.DEFINE_CHAIN_STEP ('my_chain1', 'stepB', 'my_program2'); DBMS_SCHEDULER.DEFINE_CHAIN_STEP ('my_chain1', 'stepC', 'my_program3'); END; / BEGIN DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain1', 'TRUE', 'START stepA'); DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain1', 'stepA COMPLETED', 'Start stepB, stepC'); DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END'); END; / BEGIN DBMS_SCHEDULER.ENABLE('my_chain1'); END; / BEGIN DBMS_SCHEDULER.CREATE_JOB (job_ name => 'chain_job_1', job_type => 'CHAIN', job_ action => 'my_chain1', repeat_interval => 'fre- q=daily;byhour=13;byminute=0;bysecond=0', enabled => TRUE); END; /</pre>	Create several EVENTS and manage them within a table to keep the results, or the last run status to determine when to execute the next event.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/event-scheduler.html>
- <https://dev.mysql.com/doc/refman/5.7/en/events-syntax.html>

Oracle External tables vs. MySQL Integration with Amazon S3

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Create Tables	Use Aurora MySQL integration with S3 - different paradigm and syntax

Oracle Usage

The Oracle External tables feature allows you to create a tables in a database that reads data from a source located outside the database (externally).

Beginning with Oracle 12.2, the external table can be partitioned, providing all the benefits of a regular partitioned table.

Oracle 18c adds support for inline external tables, which is a way to get data from external source in a SQL query without having to define and create external table first.

```
SELECT * FROM EXTERNAL (
(i NUMBER, d DATE)
TYPE ORACLE_LOADER
DEFAULT DIRECTORY data_dir
ACCESS PARAMETERS (
RECORDS DELIMITED BY NEWLINE
FIELDS TERMINATED BY '|'
)
LOCATION ('test.csv') REJECT LIMIT UNLIMITED
) tst_external;
```

Examples

Use CREATE TABLE with ORGANIZATION EXTERNAL to identify it as an external table. Specify the TYPE to allow the database to choose the right driver for the data source. The options are:

- ORACLE_LOADER – The data must be sourced from text data files (default).
- ORACLE_DATAPUMP –The data must be sourced from binary dump files. You can write dump files only as part of creating an external table with the CREATE TABLE AS SELECT statement. Once the dump file is created, it can be read any number of times, but it cannot be modified (that is, no DML operations can be performed).
- ORACLE_HDFS – Extracts data stored in a Hadoop Distributed File System (HDFS).
- ORACLE_HIVE – Extracts data stored in Apache HIVE.
- DEFAULT DIRECTORY – The directory database object used to find the external table.
- ACCESS PARAMETER – Defines the DELIMITER character and the query fields.
- LOCATION – The file name in the first two data source types or URI in the Hadoop data source (not in use with hive data source).

```
CREATE TABLE emp_load
(id CHAR(5), emp_dob CHAR(20), emp_lname CHAR(30), emp_fname CHAR(30), emp_start_date
DATE) ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER DEFAULT DIRECTORY data_dir ACCESS PARAMETERS
(RECORDS DELIMITED BY NEWLINE FIELDS
(id CHAR(2), emp_dob CHAR(20), emp_lname CHAR(18), emp_fname CHAR(11), emp_start_date
CHAR(10) date_format DATE mask "mm/dd/yyyy"))
LOCATION ('info.dat'));
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/18/sutil/oracle-external-tables-concepts.html#GUID-44323E01-7D72-45EC-915A-99E596769D9E>

MySQL Usage

Aurora MySQL has a capability similar to Oracle's External Tables, but requires a significant amount of syntax modifications. The main difference is that there is no open link to files and the data must be transferred from and to MySQL (if all data is needed).

There are two important operations for MySQL and S3 integration:

- Saving data to an S3 file.
- Loading data from an S3 file.

Aurora MySQL must have permissions to the S3 bucket. For more information, see the links at the end of this section.

In Oracle 18c, the inline external table feature was introduced. This cannot be achieved in Aurora for MySQL and it depends on the use case but other services can be considered. For ETLs, for example, AWS Glue can be considered.

Saving Data to S3

You can use the SELECT INTO OUTFILE S3 statement to query data from an Amazon Aurora MySQL DB cluster and save it directly into text files stored in an Amazon S3 bucket. Use this functionality to avoid transferring data to the client first, and then copying the data from the client to Amazon S3.

Note: The default file size threshold is six gigabytes (GB). If the data selected by the statement is less than the file size threshold, a single file is created. Otherwise, multiple files are created.

If the SELECT statement fails, files already uploaded to Amazon S3 remain in the specified Amazon S3 bucket. You can use another statement to upload the remaining data instead of starting over again.

If the amount of data to be selected is large (more than 25 GB), it is recommended to use multiple SELECT INTO OUTFILE S3 statements to save the data to Amazon S3.

Metadata, such as table schema or file metadata, is not uploaded by Aurora MySQL to Amazon S3.

Examples

The following statement selects all data in the employees table and saves the data into an Amazon S3 bucket in a different region from the Aurora MySQL DB cluster. The statement creates data files in which each field is ter-

minated by a comma (,) character and each row is terminated by a newline (\n) character. The statement returns an error if files that match the sample_employee_data file prefix exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-
pdx/sample_employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
```

The following statement selects all data in the employees table and saves the data into an Amazon S3 bucket in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. It also creates a manifest file. The statement returns an error if files that match the sample_employee_data file prefix exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/sample_
employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' MANIFEST ON;
```

The following statement selects all data in the employees table and saves the data into an Amazon S3 bucket in a different region from the Aurora DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. The statement overwrites any existing files that match the sample_employee_data file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-
pdx/sample_employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' OVERWRITE ON;
```

The following statement selects all data in the employees table and saves the data into an Amazon S3 bucket in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. It also creates a manifest file. The statement overwrites any existing files that match the sample_employee_data file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/sample_
employee_data'
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
MANIFEST ON OVERWRITE ON;
```

For more details, see [Saving Data from an Amazon Aurora MySQL DB Cluster into Text Files in an Amazon S3 Bucket](#)

Loading Data from S3

You can use the LOAD DATA FROM S3 or LOAD XML FROM S3 statement to load data from files stored in an Amazon S3 bucket.

Also, you can use the LOAD DATA FROM S3 statement to load data from any text file format supported by the MySQL LOAD DATA INFILE statement such as comma-delimited text data. Compressed files are not supported.

Examples

The following example runs the LOAD DATA FROM S3 statement with the manifest in the previous example (named customer.manifest). After the statement completes, an entry for each successfully loaded file is written to the aurora_s3_load_history table.

```
LOAD DATA FROM S3 MANIFEST
's3-us-west-2://aurora-bucket/customer.manifest' INTO TABLE CUSTOMER FIELDS TERMINATED
BY ',' LINES TERMINATED BY '\n'
(ID, FIRSTNAME, LASTNAME, EMAIL);
```

Each successful LOAD DATA FROM S3 statement updates the `aurora_s3_load_history` table in the `mysql` schema with an entry for each file loaded.

After you run the LOAD DATA FROM S3 statement, you can verify which files were loaded by querying the `aurora_s3_load_history` table. To see the files that were loaded from one execution of the statement, use the WHERE clause to filter the records on the Amazon S3 URI for the manifest file used in the statement. If you have used the same manifest file before, filter the results using the timestamp field.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

The following table describes the fields in the `aurora_s3_load_history` table:

Field	Description
<code>load_prefix</code>	The URI specified in the load statement. This URI can map to any of the following: <ul style="list-style-type: none"> A single data file for a LOAD DATA FROM S3 FILE statement. An Amazon S3 prefix that maps to multiple data files for a LOAD DATA FROM S3 PREFIX statement. A single manifest file containing the names of files to be loaded for a LOAD DATA FROM S3 MANIFEST statement.
<code>file_name</code>	The name of a file loaded into Aurora from Amazon S3 using the URI identified in the <code>load_prefix</code> field.
<code>version_number</code>	The version number of the file identified by the <code>file_name</code> field that was loaded (if the Amazon S3 bucket has a version number).
<code>bytes_loaded</code>	The size of the file loaded in bytes.
<code>load_timestamp</code>	The timestamp when the LOAD DATA FROM S3 statement completed.

Examples

The following statement loads data from an Amazon S3 bucket in the same region as the Aurora DB cluster. The statement reads the comma-delimited data in the file `customerdata.txt` in the `dbbucket` Amazon S3 bucket and then loads the data into the table `store-schema.customer-table`.

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv' INTO TABLE
  store-schema.customer-table FIELDS TERMINATED BY ',' LINES
  TERMINATED BY '\n' (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

The following statement loads data from an Amazon S3 bucket in a different region from the Aurora DB cluster. The statement reads the comma-delimited data from all files matching the `employee-data` object prefix in the `my-data` Amazon S3 bucket in the `us-west-2` region and then loads the data into the `employees` table.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data' INTO TABLE employees
  FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (ID, FIRSTNAME, LASTNAME, EMAIL,
  SALARY);
```

The following statement loads data from the files specified in a JSON manifest file named `q1_sales.json` into the `sales` table.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json' INTO TABLE
  sales FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (MONTH, STORE, GROSS, NET);
```

Loading XML FROM S3

You can use the `LOAD XML FROM S3` statement to load data from XML files stored on an Amazon S3 bucket in one of three different XML formats as described below.

Column names as attributes of a `<row>` element: The attribute value identifies the contents of the table field.

```
<row column1="value1" column2="value2" .../>
```

Column names as child elements of a `<row>` element: The value of the child element identifies the contents of the table field.

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

Column names in the name attribute of `<field>` elements in a `<row>` element: The value of the `<field>` element identifies the contents of the table field.

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

The following statement loads the first column from the input file into the first column of `table1` and sets the value of the `table_column2` column in `table1` to the input the value of the second column divided by 100.

```
LOAD XML FROM S3 's3://mybucket/data.xml' INTO TABLE table1 (column1, @var1) SET
table_column2 = @var1/100;
```



The following statement sets the first two columns of table1 to the values in the first two columns from the input file and then sets the value of the column3 in table1 to the current time stamp.

```
LOAD XML FROM S3 's3://mybucket/data.xml' INTO TABLE table1 (column1, column2) SET
column3 = CURRENT_TIMESTAMP;
```

You can use subqueries in the right side of SET assignments. For a subquery that returns a value to be assigned to a column, you can use only a scalar subquery. Also, you cannot use a subquery to select from the table that is being loaded.

For more details, see [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket](#)

Oracle Inline Views vs. MySQL Inline Views

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	

Oracle Usage

Inline views refer to SELECT statements located in the FROM clause of secondary SELECT statements. Inline views can help make complex queries simpler by removing compound calculations or eliminating join operations while condensing several separate queries into a single simplified query.

Examples

The SQL statement marked in red represents the inline view code. The query returns each employee matched to their salary and department id. In addition, the query returns the average salary for each department using the inline view column SAL_AVG.

```
SELECT A.LAST_NAME, A.SALARY, A.DEPARTMENT_ID, B.SAL_AVG
FROM EMPLOYEES A,
(SELECT DEPARTMENT_ID, ROUND(AVG(SALARY)) AS SAL_AVG FROM EMPLOYEES GROUP BY
DEPARTMENT_ID)
WHERE A.DEPARTMENT_ID = B.DEPARTMENT_ID;
```

MySQL Usage



MySQL semantics refer to inline views as “Subselects” or “Subqueries”. In either case, the functionality is the same. Running the Oracle inline view example in the Oracle section results in the error: “SQL Error[1248][4200]: Every derived table must have its own alias”. This error occurs because Oracle supports omission of aliases for the inner statement while in MySQL aliases are mandatory. Mandatory aliases are the only major difference when migrating Oracle inline views to MySQL.

Examples

“B” used as an alias.

```
SELECT A.LAST_NAME, A.SALARY, A.DEPARTMENT_ID, B.SAL_AVG
FROM EMPLOYEES A,
(SELECT DEPARTMENT_ID, ROUND(AVG(SALARY)) AS SAL_AVG FROM EMPLOYEES GROUP BY
DEPARTMENT_ID) B
WHERE A.DEPARTMENT_ID = B.DEPARTMENT_ID;
```

Oracle JSON Document Support vs. MySQL JSON

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		JSON	Different paradigm and syntax will require application/drivers rewrite

Oracle Usage

JSON documents are based on JavaScript syntax and allow the serialization of objects. Oracle support for JSON document storage and retrieval enables you to extend the database capabilities beyond purely relational use-cases and provides support for semi-structured data. Oracle JSON support also includes full-text search and several other functions dedicated to querying JSON documents.

Oracle 19 adds new function, `JSON_SERIALIZE`, which allows to serialize JSON objects to text.

For more information, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/adjsn/intro-to-json-data-and-oracle-database.html#GUID-17642E43-7D87-4590-8870-06E9FDE9A6E9>.

Examples

Create a table to store a JSON document in a data column and insert a JSON document into the table.

```
CREATE TABLE json_docs (id RAW(16) NOT NULL, data CLOB,
CONSTRAINT json_docs_pk PRIMARY KEY (id),
CONSTRAINT json_docs_json_chk CHECK (data IS JSON));

INSERT INTO json_docs (id, data) VALUES (SYS_GUID(),
'{'
  "FName"      : "John",
  "LName"      : "Doe",
  "Address"    : {
    "Street"    : "101 Street",
    "City"      : "City Name",
    "Country"   : "US",
    "Pcode"    : "90210"}
}');
```

Unlike XML data, which is stored using the SQL data type `XMLType`, JSON data is stored in an Oracle Database using the SQL data types `VARCHAR2`, `CLOB`, and `BLOB`. Oracle recommends that you always use an `is_json` check constraint to ensure the column values are valid JSON instances. Or, add a constraint at the table-level (`CONSTRAINT json_docs_json_chk CHECK (data IS JSON)`).

You can query a JSON document directly from an SQL query without using special functions. Querying without functions is called Dot Notation.

```
SELECT a.data.FName, a.data.LName, a.data.Address.Pcode AS Postcode
FROM json_docs a;
```

```
FNAME      LNAME      POSTCODE
-----
John       Doe        90210
```

```
1 row selected.
```

In addition, Oracle provides multiple SQL functions that integrate with the SQL language and enable querying JSON documents (such as IS JSON, JSON_VALUE, JSON_EXISTS, JSON_QUERY, and JSON_TABLE).

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/adjsn/intro-to-json-data-and-oracle-database.html#GUID-17642E43-7D87-4590-8870-06E9FDE9A6E9>

MySQL Usage

Aurora MySQL 5.7 supports a native JSON data type for storing JSON documents, which provides several benefits over storing the same document as a generic string. All JSON documents stored as a JSON data type are validated for correctness. If the document is not valid JSON, it is rejected and an error condition is raised. In addition, more efficient storage algorithms enable optimized read access to elements within the document. The optimized internal binary representation of the document enables much faster operation on the data without requiring expensive re-parsing.

For example:

```
CREATE TABLE JSONTable (DocumentIdentifier INT NOT NULL PRIMARY KEY, JSONDocument
JSON);
```

MySQL 5.7.22 also added the JSON utility function JSON_PRETTY() which outputs an existing JSON value in an easy-to-read format; each JSON object member or array value is printed on a separate line and a child object or array is indented 2 spaces with respect to its parent. This function also works with a string that can be parsed as a JSON value. For more detailed information and examples see [Section 12.18.8 JSON Utility Functions](#).

MySQL 5.7.22 also added the JSON utility functions JSON_STORAGE_SIZE() and JSON_STORAGE_FREE(). JSON_STORAGE_SIZE() returns the storage space in bytes used for the binary representation of a JSON document prior to any partial update.

JSON_STORAGE_FREE() shows the amount of space freed after it has been partially updated using JSON_SET() or JSON_REPLACE(); this is greater than zero if the binary representation of the new value is less than that of the previous value. Each of these functions also accepts a valid string representation of a JSON document. For such a value JSON_STORAGE_SIZE() returns the space used by its binary representation following its conversion to a JSON document. For a variable containing the string representation of a JSON document JSON_STORAGE_FREE() returns zero. Either function produces an error if its (non-null) argument cannot be parsed as a valid JSON document and NULL if the argument is NULL. For more information and examples see [Section 12.18.8 JSON Utility Functions](#).

RDS MySQL Only (not Aurora MySQL): MySQL 8 added two JSON aggregation functions JSON_ARRAYAGG() and JSON_OBJECTAGG(). JSON_ARRAYAGG() takes a column or expression as its argument and aggregates the result as a single JSON array. The expression can evaluate to any MySQL data type; this does not have to be a JSON value. JSON_OBJECTAGG() takes two columns or expressions which it interprets as a key and a value; it returns the result as a single JSON object. For more information and examples see [Section 12.20 Aggregate Functions](#)

RDS MySQL Only (not Aurora MySQL): MySQL 8.0.17 adds two functions JSON_SCHEMA_VALID() and JSON_SCHEMA_VALIDATION_REPORT() for validating JSON documents. JSON_SCHEMA_VALID() returns TRUE (1) if the document validates against the schema and FALSE (0)

if it does not. `JSON_SCHEMA_VALIDATION_REPORT()` returns a JSON document containing detailed information about the results of the validation.

JSON Functions

Aurora MySQL supports a rich set of more than 25 targeted functions for working with JSON data. These functions enable adding, modifying, and searching JSON data. Additionally, spatial JSON functions can be used for GeoJSON documents. For more information, see <https://dev.mysql.com/doc/refman/5.7/en/spatial-geojson-functions.html>.

The `JSON_ARRAY`, `JSON_OBJECT`, and `JSON_QUOTE` functions all return a JSON document from a list of values, a list of key-value pairs, or a JSON value respectively.

For example:

```
SELECT JSON_OBJECT('Person', 'John', 'Country', 'USA');

{"Person": "John", "Country": "USA"}
```

The `JSON_CONTAINS`, `JSON_CONTAINS_PATH`, `JSON_EXTRACT`, `JSON_KEYS`, and `JSON_SEARCH` functions are used to query and search the content of a JSON document.

The `CONTAINS` functions are Boolean functions that return 1 or 0 (TRUE or FALSE). `JSON_EXTRACT` returns a subset of the document based on the XPATH expression.

`JSON_KEYS` returns a JSON array consisting of the top-level key (or path top level) values of a JSON document. The `JSON_SEARCH` function returns the path to one or all of the instances of the search string.

Examples

```
SELECT JSON_EXTRACT(['Mary', 'Paul', ['Jim', 'Ryan']], '$[1]');
"Paul"

SELECT JSON_SEARCH(['Mary', 'Paul', ['Jim', 'Ryan']], 'one', 'Paul');
"$[1]"
```

Aurora MySQL supports the following functions for adding, deleting, and modifying JSON data, :

`JSON_INSERT`, `JSON_REMOVE`, `JSON_REPLACE`, and their `ARRAY` counterparts, which are used to create, delete, and replace existing data elements. For example:

```
SELECT JSON_ARRAY_INSERT(['Mary', 'Paul', 'Jim'], '$[1]', 'Jack');

["Mary", "Jack", "Paul", "Jim"]
```

`JSON_SEARCH` is used to find the location of an element value within a JSON document. For example:

```
SELECT JSON_SEARCH(['Mary', 'Paul', ['Jim', 'Ryan']], 'one', 'Paul');

"$[1]"
```

JSON Indexes

JSON columns are effectively a BINARY family type, which cannot be indexed. As an alternative, use CREATE TABLE or ALTER TABLE to add generated columns that represent some value from the JSON document and create an index on the generated column. For more information, see [Generated Columns](#).

Note: If indexes on generated columns exist for JSON documents, the query optimizer can use them to match JSON expressions and optimize data access.

Summary



The following table identifies similarities, differences, and key migration considerations.

Feature	Oracle	Aurora MySQL
JSON functions	IS_JSON, IS_NOT_JSON, JSON_EXISTS, JSON_VALUE, JSON_QUERY, JSON_TABLE	A set of more than 25 dedicated JSON functions. See https://dev.mysql.com/doc/refman/5.7/en/json-function-reference.html
Return the full JSON document / all JSON documents	emp_data is a column that stores json documents: SELECT emp_data FROM employees;	emp_data is a column that stores json documents: SELECT emp_data FROM employees;
Return a specific element from a JSON document	Return only the address property: SELECT e.emp_data.address FROM employees e;	Return only the address property: SELECT emp_data->"\$.address" FROM employees;
Return JSON documents matching a pattern in any field	Return the JSON based on a search of on all JSON properties. Could be returned even if element is equal to the pattern SELECT e.emp_data FROM employees e WHERE e.emp_data like '%patten%';	Return the JSON based on a search of on all JSON properties. Could be returned even if element is equal to the pattern SELECT e.emp_data FROM employees e WHERE e.emp_data like '%patten%';
Return JSON documents matching a pattern in specific fields (root level)	SELECT e.emp_data.name FROM employees e WHERE e.data.active = 'true';	SELECT emp_data.name FROM employees WHERE emp_data->"\$.active" = 'true';

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/json.html>
- <https://dev.mysql.com/doc/refman/5.7/en/json-functions.html>

Oracle Materialized Views vs. MySQL Summary Tables or Views

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Materialized Views	MySQL doesn't support a materialized VIEW

Oracle Usage

Oracle Materialized Views (also known as MViews) are table segments where the contents are periodically refreshed based on the results of a stored query. Oracle Materialized Views are defined with specific queries and can be manually or automatically refreshed based on specific configurations. A Materialized View runs its associated query and stores the results as a table segment.

Oracle Materialized Views are especially useful for:

- Replication of data across multiple databases.
- Data warehouse use cases.
- Increasing performance by persistently storing the results of complex queries as database tables.

Like ordinary views, Materialized Views are created with a SELECT query. The FROM clause of an MView query can reference tables, views, and other Materialized Views. The source objects that an Mview uses as data sources are also called “master tables” (replication terminology) or “detail tables” (data warehouse terminology).

Immediate vs. Deferred Refresh

When creating Materialized Views, the BUILD IMMEDIATE option can be specified to instruct Oracle to immediately update the contents of the Materialized View by running the underlying query. This is different from a deferred update where the Materialized View is populated only on the first requested refresh.

Fast and Complete Refresh

- **REFRESH FAST:** Incremental data refresh. Only updates rows that have changed since the last refresh of the Materialized View instead of performing a complete refresh. This type of refresh fails if Materialized View Logs have not been created.
- **COMPLETE:** The table segment used by the Materialized View is truncated (data is cleared) and repopulated by running the associated query.

Materialized View Logs

When creating Materialized Views, a Materialized View Log can be used to instruct Oracle to store any changes performed by DML commands on the “master tables” that are used to refresh the Materialized View, which provides faster Materialized View refreshes. Without Materialized View Logs, Oracle must re-execute the query associated with the Materialized View each time (also known as a “complete refresh”). This process is slower compared to using Materialized View Logs.

Materialized View Refresh Strategy

- **ON COMMIT:** Refreshes the Materialized View upon any commit made on the underlying associated tables.
- **ON DEMAND:** The refresh is initiated via a scheduled task or manually by the user.

Examples

Create a simple Materialized View named mv1 that executes a simple SELECT statement on the employees table.

```
CREATE MATERIALIZED VIEW mv1 AS SELECT * FROM hr.employees;
```

Create a more complex Materialized View using a Database Link (remote) to obtain data from a table located in a remote database. This Materialized View also contains a subquery. The FOR UPDATE clause allows the Materialized View to be updated.

```
CREATE MATERIALIZED VIEW foreign_customers FOR
UPDATE AS SELECT * FROM sh.customers@remote cu WHERE EXISTS
(SELECT * FROM sh.countries@remote co WHERE co.country_id = cu.country_id);
```

Create a Materialized View on two source tables: times and products. This approach enables FAST refresh of the Materialized View instead of the slower COMPLETE refresh. Also, create a new Materialized View named sales_mv which is refreshed incrementally (REFRESH FAST) each time changes in data are detected (ON COMMIT) on one or more of the tables associated with the Materialized View query.

```
CREATE MATERIALIZED VIEW LOG ON times
WITH ROWID, SEQUENCE (time_id, calendar_year)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON products
WITH ROWID, SEQUENCE (prod_id)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW sales_mv
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS SELECT t.calendar_year, p.prod_id,
SUM(s.amount_sold) AS sum_sales
FROM times t, products p, sales s
WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
GROUP BY t.calendar_year, p.prod_id;
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/dwhsg/basic-materialized-views.html#GUID-A7AE8E5D-68A5-4519-81EB-252EAAF0ADFF>

MySQL Usage

Oracle Materialized Views have no equivalent feature in MySQL, but other features can be used separately or combined to achieve similar functionality.

You must evaluate each case on its own merits, but options include:

- **Summary Tables:** If your MView has many calculations and data manipulations, you can keep the results in tables and query the data without running all calculations on-the-fly. The data for these tables can be copied using triggers or events objects.
- **Views:** Aurora MySQL has a new Parallel Query mechanism that offloads some of the query operations to the storage level. This approach can greatly improve performance. In some cases, regular Views can be used and may decrease some administration tasks. To evaluate this option, measure the performance and execution time of your SQL.

For more details, see:

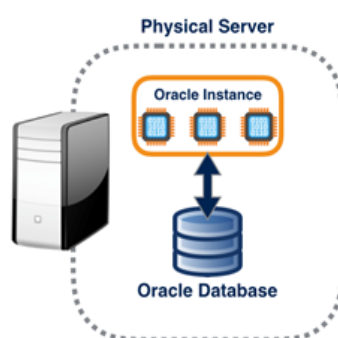
- <https://dev.mysql.com/doc/refman/5.7/en/create-table.html>
- <https://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html>
- <https://dev.mysql.com/doc/refman/5.7/en/create-view.htmlx>

Oracle Multitenant vs. MySQL Databases

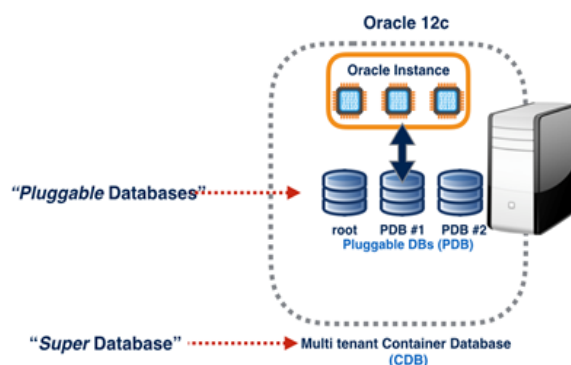
Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Distribute load/applications/ users across multiple instances

Oracle Usage

Oracle 12c introduces a new multitenant architecture that provides the ability to create additional independent pluggable databases under a single Oracle instance. Prior to Oracle 12c, a single Oracle database instance only supported running a single Oracle database as shown in the diagram below.



Oracle 12c introduces a new multi-Container Database (CBD) that supports one or more Pluggable Databases (PDB). The CBD can be thought of as a single superset database with multiple pluggable databases. The relationship between an Oracle instance and databases is now 1:N.



Oracle 18c adds following multi-tenant related features:

- DBCA PDB Clone: UI interface which allows cloning multiple pluggable databases (PDB).
- Refreshable PDB Switchover: ability to switch roles between pluggable database clone and its original master.

- CDB Fleet Management: ability to group multiple container databases (CDB) into fleets that can be managed as a single logical database.

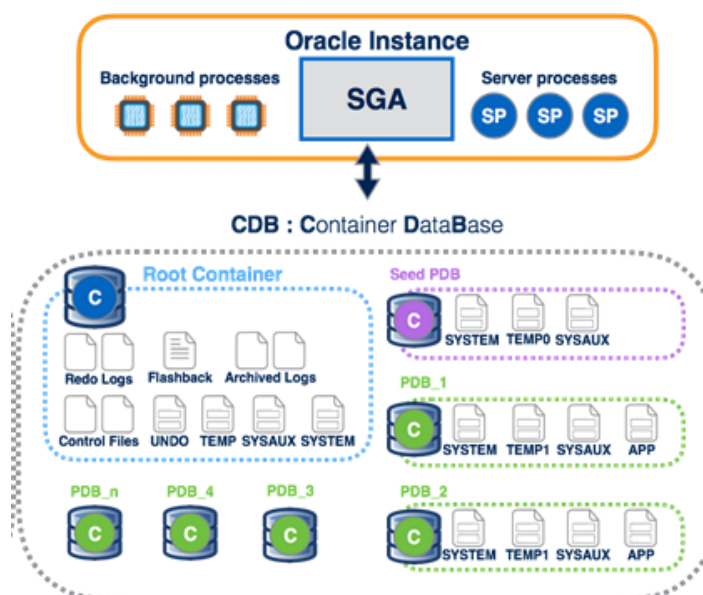
Oracle 19 introduced support to having more than one pluggable database (PDB) in a container database (CDB) in sharded environments.

Advantages of the Oracle 12c Multi-tenant Architecture

- PDBs can be used to isolate applications from one another.
- PDBs can be used as portable collection of schemas.
- PDBs can be cloned and transported to different CDBs/Oracle instances.
- Management of many databases (individual PDBs) as a whole.
- Separate security, users, permissions, and resource management per PDB provides greater application isolation.
- Enables a consolidated database model of many individual applications sharing a single Oracle server.
- Provides an easier way to patch and upgrade individual clients and/or applications using PDBs.
- Backups are supported at both a multitenant container-level as well as at an individual PDB-level (both for physical and logical backups).

The Oracle multitenant architecture

- A multitenant CDB can support one or more PDBs.
- Each PDB contains its own copy of SYSTEM and application tablespaces.
- The PDBs share the Oracle Instance memory and background processes. The use of PDBs enables consolidation of many databases and applications into individual containers under the same Oracle instance.
- A single Root Container (CDB\$ROOT) exists in a CDB and contains the Oracle Instance Redo Logs, undo tablespace (unless Oracle 12.2 local undo mode is enabled), and control files.
- A single Seed PDB exists in a CDB and is used as a template for creating new PDBs.



CDB and PDB Semantics

- CDB
 - Created as part of the Oracle 12c software installation.
 - Contains the Oracle control files, its own set of system tablespaces, the instance undo tablespaces (unless Oracle 12.2 local undo mode is enabled), and the instance redo logs.
 - Holds the data dictionary for the root container and for all of the PDBs.
- PDB
 - An independent database that exists under a CDB. Also known as a "container".
 - Used to store application-specific data.
 - Can be created from a the `pdb$seed` (template database) or as a clone of an existing PDB.
 - Stores metadata information specific to its own objects (data-dictionary).
 - Has its own set of application data files, system data files, and tablespaces along with temporary files to manage objects.

Examples

List existing PDBs created in an Oracle CDB instance.

```
SHOW PDBS;
CON_ID          CON_NAME          OPEN MODE  RESTRICTED
-----
2              PDB$SEED         READ ONLY  NO
3              PDB1             READ WRITE NO
```

Provision a new PDB from the template `seed$pdb`.

```
CREATE PLUGGABLE DATABASE PDB2 admin USER ora_admin IDENTIFIED BY
ora_admin FILE_NAME_CONVERT=('/pdbseed/', '/pdb2/');
```

Alter a specific PDB to READ/WRITE and verify.

```
ALTER PLUGGABLE DATABASE PDB2 OPEN READ WRITE;
SHOW PDBS;

CON_ID          CON_NAME          OPEN MODE  RESTRICTED
-----
2              PDB$SEED         READ ONLY  NO
3              PDB1             READ WRITE NO
4              PDB2             READ WRITE NO
```

Clone a PDB from an existing PDB.

```
CREATE PLUGGABLE DATABASE PDB3 FROM PDB2 FILE_NAME_CONVERT= ('/pdb2/', '/pdb3/');
SHOW PDBS;

CON_ID          CON_NAME          OPEN MODE  RESTRICTED
-----
2              PDB$SEED         READ ONLY  NO
3              PDB1             READ WRITE NO
```

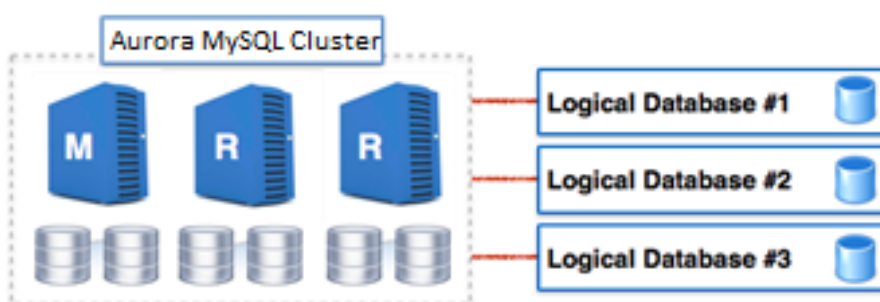
4	PDB2	READ WRITE	NO
5	PDB3	MOUNTED	

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/multi/index.html>

MySQL Usage

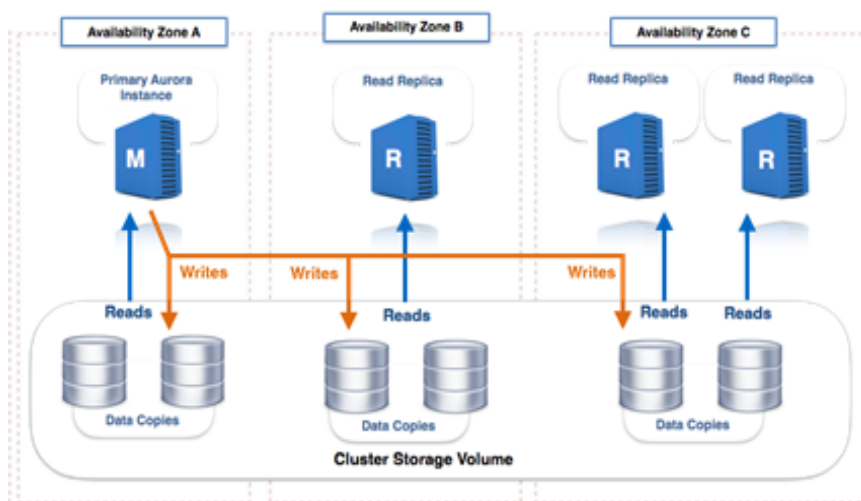
Amazon Aurora MySQL offers a different and simplified architecture to manage and create a multitenant database environment. Using Aurora MySQL, it is possible to provide levels of functionality similar (but not identical) to those offered by Oracle PDBs by creating multiple databases under the same Aurora MySQL cluster and/or using separate Aurora clusters if total isolation of workloads is required.

Multiple MySQL databases can be created under a single Amazon Aurora MySQL Cluster.



Each Amazon Aurora cluster contains a primary instance that can accept both reads and writes for all cluster databases.

Up to 15 read-only nodes can be created providing scale-out functionality for application reads and high availability.



An Oracle CDB/Instance is a high-level equivalent to an Amazon Aurora cluster, and an Oracle Pluggable Database (PDB) is equivalent to a MySQL database created inside the Amazon Aurora cluster. Not all features are comparable between Oracle 12c PDBs and Amazon Aurora.

Starting at Oracle 18c and 19c , a new usages of this feature were introduces like:

- PDB Clone
- Refreshable PDB Switchover
- CDB Fleet Management
- More than one pluggable database (PDB) in a container database (CDB) in sharded environments.

In the AWS cloud, these features can be achieved in many ways and each can be optimized using different services.

Cloning a databases inside the MySQL instance is not so easy, for the same instance, export/import can be used.

To achieve simliar functionality like Refreshable PDB Switchover, it depedns on the use case but there are multiple options mostly depended on the required granularity:

- Databases in the same instance - "failover" can be done using CREATE DATABASE stamtement when size and required duration allow that and use an application failover to point to any of the databases.
- Database links and replication method - database links or AWS DMS can be used to make sure there are two databases in two different instances that are in sync and have application failover to point to the other database when needed.

Managing CDB is actually very similar to the AWS orchestration, as you can manage multiple RDS instances there (CDB) and databases inside (PDB), all monitored centrally and can be managed through the AWS console or AWS CLI.

Examples

Create a new database in MySQL using the CREATE DATABASE statement.

```
CREATE DATABASE my_db1;
CREATE DATABASE my_db2;
CREATE DATABASE my_db3;
```

List all databases created under an Amazon Aurora MySQL cluster.

```
SHOW DATABASES;
```

```
Database
-----|
information_schema |
mysql              |
performance_schema |
my_db1             |
my_db2             |
my_db3             |
sys                |
tmp                |
```

Independent Database Backups


Oracle 12c provides the ability to perform both logical backups (via DataPump) and physical backups (via RMAN) at both the CDB and PDB levels. Similarly, Amazon Aurora MySQL provides the ability to perform logical backups on all or a specific database(s) using mysqldump. However, for physical backups when using snapshots,

the entire cluster and all databases are included in the snapshot. Backing up a specific database within the cluster is not supported.

This is usually not a concern because volume snapshots are extremely fast operations that occur at the storage infrastructure layer, incur minimal overhead, and operate at extremely fast speeds. However, the process of restoring a single MySQL database from an Aurora snapshot requires additional steps such as exporting the specific database after a snapshot restore and importing it back to the original Aurora cluster.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/create-database.html>

Oracle Resource Manager vs. MySQL Dedicated Amazon Aurora Clusters

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Distribute load/applications/ users across multiple instances

Oracle Usage

Oracle's Resource Manager enables enhanced management of multiple concurrent workloads running under a single Oracle database. Using Oracle Resource Manager, you can partition server resources for different workloads. Resource Manager helps with sharing server and database resources without causing excessive resource contention and helps to eliminate scenarios involving inappropriate allocation of resources across different database sessions.

Oracle Resource Manager enables you to:

- Guarantee a minimum amount of CPU cycles for certain sessions regardless of other running operations.
- Distribute available CPU by allocating percentages of CPU time to different session groups.
- Limit the degree of parallelism of any operation performed by members of a user group.
- Manage the order of parallel statements in the parallel statement queue.
- Limit the number of parallel execution servers that a user group can use.
- Create an active session pool. An active session pool consists of a specified maximum number of user sessions allowed to be concurrently active within a user group.
- Monitor used database/server resources by dictionary views.
- Manage runaway sessions or calls and prevent them from overloading the database.
- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit.
- Limit the amount of time a session can be connected but idle, thus forcing inactive sessions to disconnect and potentially freeing memory resources.
- Allow a database to use different resource plans, based on changing workload requirements.
- Manage CPU allocation when there is more than one instance on a server in an Oracle Real Application Cluster environment (also called instance caging).

Oracle Resource Manager introduces three concepts:

Consumer Group: A collection of sessions grouped together based on resource requirements. The Oracle Resource Manager allocates server resources to resource consumer groups, not to the individual sessions.

Resource Plan: Specifies how the database allocates its resources to different Consumer Groups. You must specify how the database allocates resources by activating a specific resource plan.

Resource Plan Directive: Associates a resource consumer group with a plan and specifies how resources are allocated to that resource consumer group.

Notes:

- Only one Resource Plan can be active at any given time.
- Resource Directives control the resources allocated to a Consumer Group that belongs to a Resource Plan.
- The Resource Plan can refer to Subplans to create even more complex Resource Plans.

Examples

Create a Simple Resource Plan. To enable the Oracle Resource Manager, you must assign a plan name to the RESOURCE_MANAGER_PLAN parameter. Using an empty string disables the Resource Manager.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

Create a complex Resource Plan. A complex Resource Plan is a plan not created with the CREATE_SIMPLE_PLAN PL/SQL procedure. It provides more flexibility and granularity.

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
PLAN          => 'DAYTIME',
GROUP_OR_SUBPLAN => 'OLTP',
COMMENT       => 'OLTP group',
MGMT_P1       => 75);
END;
/
```

For additional details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-resources-with-oracle-database-resource-manager.html#GUID-2BEF5482-CF97-4A85-BD90-9195E41E74EF>

MySQL Usage

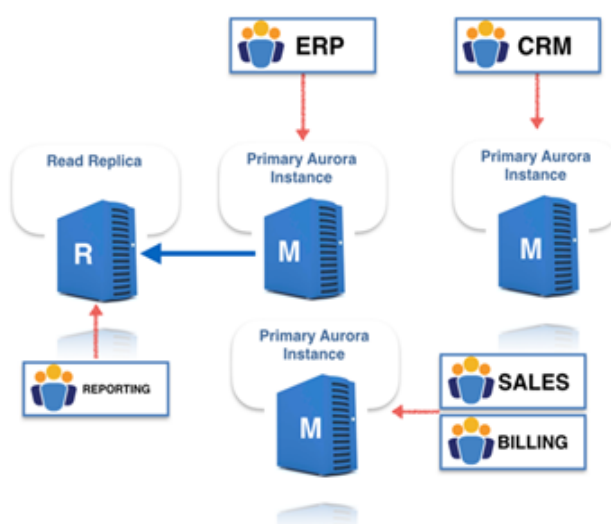
MySQL does not have built-in resource management capabilities that are equivalent to the functionality provided by Oracle Resource Manager. However, due to the elasticity and flexibility provided by “cloud economics”, work-arounds could be applicable and such capabilities might not be as of similar importance to monolithic on-premises databases.

The Oracle Resource Manager primarily exists because, traditionally, Oracle databases were installed on very powerful monolithic servers that powered multiple applications simultaneously. The monolithic model made the most sense in an environment where the licensing for the Oracle database was per CPU and where Oracle databases were deployed on physical hardware. In these scenarios, it made sense to consolidate as many workloads as possible into few servers. In cloud databases, the strict requirement to maximize the usage of each individual “server” is often not as important and a different approach can be employed.

Individual Amazon Aurora clusters can be deployed, with varying sizes, each dedicated to a specific application or workload. Additional read-only Aurora Replica servers can be used to offload any reporting-style workloads from the master instance.



The traditional Oracle model where maximizing the usage of each physical Oracle server was essential due to physical hardware constraints and the per-CPU core licensing model.



With Amazon Aurora, separate and dedicated database clusters can be deployed, each dedicated to a specific application/workload creating isolation between multiple connected sessions and applications.

Each Amazon Aurora instance (Primary/Replica) can scale independently in terms of CPU and memory resources using the different “instance types”. Because multiple Amazon Aurora Instances can be instantly deployed and much less overhead is associated with the deployment and management of Aurora instances when compared to physical servers, separating different workloads to different instance classes could be a suitable solution for controlling resource management.

For more information about instance types and resources, see [Amazon EC2 Instance Types](#).

In addition, each Amazon Aurora primary/replica instance can also be directly accessed from your applications using its own endpoint. This capability is especially useful if you have multiple Aurora read-replicas for a given cluster and you wish to utilize different Aurora replicas to segment your workload.

Examples

Suppose that you were using a single Oracle Database for multiple separate applications and used Oracle Resource Manager to enforce a workload separation, allocating a specific amount of server resources for each application. With Amazon Aurora, you might want to create multiple separate databases for each individual application. Adding additional replica instances to an existing Amazon Aurora cluster is easy.

1. Navigate to [instances section under RDS](#).
2. Select the Amazon Aurora cluster to scale-out by adding an additional read Replica. Click the **Actions** button and then click **Add reader**.

The screenshot shows the AWS RDS Databases console. A table lists several database instances. The 'pg-playbooks' instance is selected, and the 'Actions' menu is open, showing options like 'Add reader', 'Create cross-Region read replica', and 'Create clone'. The 'Add reader' option is highlighted with a red '3', and the 'Actions' button is labeled with a red '2'.

DB identifier	Role	Engine	Region & AZ	Size	Status
mysql-aurora-playbook	Regional	Aurora MySQL	eu-central-1	1 instance	Available
mysql-aurora-playbook-instance-1	Writer	Aurora MySQL	eu-central-1a	db.t3.medium	Available
oraplaybook	Instance	Oracle Standard Edition Two	eu-central-1a	db.t3.medium	Available
pg-playbooks 1	Regional	Aurora PostgreSQL	eu-central-1	1 instance	Available
pg-playbooks-instance-1	Writer	Aurora PostgreSQL	eu-central-1a	db.t3.medium	Available

3. Select the instance class depending on the amount of compute resources your application requires.

DB instance class

DB instance class [Info](#)
 Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Memory optimized classes (includes r classes)
 Burstable classes (includes t classes)

db.r5.large	2 vCPUs	16 GiB RAM	Network: 4,750 Mbps
db.r5.large	2 vCPUs	16 GiB RAM	Network: 4,750 Mbps
db.r5.xlarge	4 vCPUs	32 GiB RAM	Network: 4,750 Mbps
db.r5.2xlarge	8 vCPUs	64 GiB RAM	Network: 4,750 Mbps
db.r5.4xlarge	16 vCPUs	128 GiB RAM	Network: 4,750 Mbps
db.r5.8xlarge	32 vCPUs	256 GiB RAM	Network: 6,800 Mbps
db.r5.12xlarge	48 vCPUs	384 GiB RAM	Network: 9,500 Mbps
db.r5.16xlarge	64 vCPUs	512 GiB RAM	Network: 13,600 Mbps
db.r5.24xlarge	96 vCPUs	768 GiB RAM	Network: 19,000 Mbps
db.r6g.large	2 vCPUs	16 GiB RAM	Network: 4,750 Mbps

Virtual private cloud (VPC) [Info](#)



recommended for

4. When completed, click **Create Aurora Replica**.

Summary

Oracle Resource Manager	Amazon Aurora Instances
Set the maximum CPU usage for a resource group	Create a dedicated Aurora Instance for a specific application.
Limit the degree of parallelism for specific queries	N/A
Limit parallel execution	N/A
Limit the number of active sessions	<p>Manually detect the number of connections open from a specific application and restrict connectivity either via database procedures or within the application Data Access Layer (DAL) itself.</p> <pre>select count(*) from information_schema.processlist where user='USER_NAME' and COMMAND<>'Sleep';</pre>
Restrict maximum runtime of queries	SET max_execution_time TO X;
Limit the maximum idle time for sessions	<p>Manually detect the number of connections open from a specific application and restrict connectivity either via database procedures or within the application DAL itself.</p> <pre>select count(*) from information_schema.processlist where user='USER_NAME' and COMMAND='Sleep' and TIME > X;</pre>
Limit the time that an idle session holding open locks can block other sessions	<p>Manually detect the number of connections open from a specific application and restrict connectivity either via database procedures or within the application DAL itself.</p> <pre>select count(*) from information_schema.processlist where user='USER_NAME' and COMMAND='Sleep';</pre>
Use “instance caging” in a multi-node Oracle RAC Environment	Similar capabilities can be achieved by separating different applications to different Aurora clusters or, for read-only workloads, separate Aurora read replicas within the same Aurora cluster.

Oracle SecureFile LOBs vs. MySQL Large Objects (LOB)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	SecureFiles are not supported, automation and compatibility refer only to LOBs

Oracle Usage

Large Objects (LOB) is a mechanism for storing binary data in a database. Oracle 11g introduced Secure File LOBS that provide more efficient storage. They are created using the SECUREFILE keyword as part of the CREATE TABLE statement.

The Primary benefits of using SECUREFILE lobs are:

- **Compression:** Uses Oracle advanced compression to analyze SecureFiles LOB data to save disk space.
- **De-Duplication:** Automatically detects duplicate LOB data within a LOB column or partition and reduces storage space by removing duplicates of repeating binary data.
- **Encryption:** Combined with Transparent Data Encryption (TDE).

Examples

Create a table using a SecureFiles LOB column.

```
CREATE TABLE sf_tab (COL1 NUMBER, COL2_CLOB CLOB)
LOB(COL2_CLOB) STORE AS SECUREFILE;
```

Provide additional options for LOB compression during table creation.

```
CREATE TABLE sf_tab (COL1 NUMBER, COL2_CLOB CLOB)
LOB(COL2_CLOB) STORE AS SECUREFILE COMPRESS_LOB (COMPRESS HIGH);
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/adlob/introduction-to-large-objects.html#GUID-1A2B0023-9EE8-48AF-AA76-171D1FC5C241>

MySQL Usage

MySQL does not support the advanced storage, security, and encryption options of Oracle SecureFile LOBs. Regular Large Objects data types are supported by MySQL and provide stream-style access.

The four Binary Large Object (BLOB) types are:

- TINYBLOB
- BLOB

- MEDIUMBLOB
- LONGBLOB.

These types differ only in the maximum length of the values they can hold.

The four TEXT types are:

- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

BLOB values are treated as binary strings (byte strings). They have the binary character set, collation, and comparison. Sorting is based on the numeric values of the bytes in column values.

TEXT values are treated as non-binary strings (character strings). They have a character set other than binary. Values are sorted and compared based on the collation of the character set.

For TEXT columns, index entries are space-padded at the end. If the index requires unique values, duplicate-key errors occur for values that differ only in the number of trailing spaces. For example, if a table contains 'b', an attempt to store 'b ' causes a duplicate-key error.

Because BLOB and TEXT values can be extremely long, there are some constraints:

- Only the first *max_sort_length* bytes (default is 1024) of the column are used when sorting. You can make more bytes significant in sorting or grouping by increasing its value at server startup or runtime. Clients can change the value of this variable
- BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types. Use of disk incurs a performance penalty. Therefore, include BLOB or TEXT columns in the query result only if they are essential.
- BLOB or TEXT types determine the maximum size, but the largest value that can be transmitted between the client and server is determined by the amount of available memory and the size of the communications buffers. Message buffer size can be changed by the *max_allowed_packet* variable, but it must be done for both server and client.



Examples

Create a table using a BLOB column with an index.

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/create-table.html> and <https://dev.mysql.com/doc/refman/5.7/en/blob.html>

Oracle Synonyms vs. MySQL Synonyms

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		SCT Action Codes - Synonyms	Use stored procedures and functions to abstract instance-wide objects

Oracle Usage

Synonyms are database objects that serve as alternative identifiers for other database objects. The referenced database object is called the 'base object' and may reside in the same database, another database on the same instance, or on a remote server.

Synonyms provide an abstraction layer to isolate client application code from changes to the name or location of the base object.

In Oracle, Synonyms are often used to simplify the object's name to avoid referring to the other schema as well as for security reasons.

For example, table A resides in schema A, and the client application accesses it through a Synonym. Table A needs to be moved to another schema. To make the move seamless, only the Synonym definition should be updated. Without Synonyms, the client application code must be rewritten to access the other schema or to change the connection string. Instead, you can create a Synonym called Table A and it will transparently redirect the calling application to the new schema without any code changes.

Synonyms can be created for the following objects:

- Assembly (CLR) stored procedures, table-valued functions, scalar functions, and aggregate functions
- Stored procedures and functions
- User defined tables including local and global temporary tables
- Views

Syntax

```
CREATE [OR REPLACE] [EDITIONABLE | NONEDITIONABLE] [PUBLIC] SYNONYM [schema .] synonym_name FOR [schema .] object_name [@ dblink];
```

Note: The EDITIONABLE | NONEDITIONABLE options are for to determine if this object will be private or public. For more information, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/editions.html#GUID-F0D940E0-618D-4656-982E-1C5E49FCCD42>.

Examples

The command below will create synonym object local_emp referring to the usa.emp table:

```
CREATE SYNONYM local_emp FOR usa.emp;
```

Note: After running the command above, when referring to local_emp, the command or query will be executed against usa.emp.

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-SYNONYM.html#GUID-A806C82F-1171-478E-A910-F9C6C42739B2>

MySQL Overview



Synonyms are not supported in Aurora MySQL and there is no known generic workaround.

A partial workaround is to use encapsulating views as an abstraction layer for accessing tables (or views). Similarly, you can also use functions or stored procedures that call other functions or stored procedures.

Note: Synonyms are often used in conjunction with Database Links, which are not supported by Aurora MySQL.

For more information, see [DB Links](#), [Views](#), [Functions](#), and [Stored Procedures](#).

Oracle Views vs. MySQL Views

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Views	<p>MySQL does not support View with READ ONLY option</p> <p>MySQL does not support DML on views</p>

Oracle Usage

Database Views store a named SQL query in the Oracle Data Dictionary with a predefined structure. A view does not store actual data and may be considered a “virtual table” or a “logical table” based on the data from one or more “physical” database tables.

Privileges

- A user must have the CREATE VIEW privilege to create a view in their own schema.
- A user must have the CREATE ANY VIEW privilege to create a view in any schema.
- The owner of a view must have all the necessary privileges on the source tables or views on which the view is based (SELECT or DML privileges).

CREATE (OR REPLACE) VIEW Statements

- **CREATE VIEW:** Creates a new view.
- **CREATE OR REPLACE:** Overwrites an existing view and modifies the view definition without having to manually drop and recreate the original view, and without deleting the previously granted privileges.

Oracle Common View Parameters

Oracle View Parameter	Description
CREATE OR REPLACE	Recreate an existing view (if one exists) or create a new view.
FORCE	Create the view regardless of the existence of the source tables or views and regardless of view privileges.
VISIBLE INVISIBLE	Specify if a column based on the view is visible or invisible.
WITH READ ONLY	Disable DML commands.
WITH CHECK OPTION	Specifies the level of enforcement when performing DML commands on the view.

Examples

Views are classified as either Simple or Complex.

A Simple View is a view having a single source table with no aggregate functions.

DML operations can be performed on simple views and affect the base table(s). The following example creates and updates a Simple View.

```
CREATE OR REPLACE VIEW VW_EMP
AS
SELECT EMPLOYEE_ID, LAST_NAME, EMAIL, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID BETWEEN 100 AND 130;
UPDATE VW_EMP
SET EMAIL=EMAIL||'.org'
WHERE EMPLOYEE_ID=110;

1 rows updated.
```

A Complex View is a view with several source tables or views containing joins, aggregate (group) functions, or an order by clause. Performing DML operations on complex views cannot be done directly, but INSTEAD OF triggers can be used as a workaround. The following example creates and updates a Complex view.

```
CREATE OR REPLACE VIEW VW_DEP
AS
SELECT B.DEPARTMENT_NAME, COUNT(A.EMPLOYEE_ID) AS CNT
FROM EMPLOYEES A JOIN DEPARTMENTS B USING(DEPARTMENT_ID)
GROUP BY B.DEPARTMENT_NAME;
UPDATE VW_DEP
SET CNT=CNT +1
WHERE DEPARTMENT_NAME=90;

ORA-01732: data manipulation operation not legal on this view
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-VIEW.html#GUID-61D2D2B4-DACC-4C7C-89EB-7E50D9594D30>

MySQL Usage

Similar to Oracle, Aurora MySQL views consist of a SELECT statement that references base tables and other views.

Aurora MySQL views are created using the CREATE VIEW statement. The SELECT statement comprising the definition of the view is evaluated only when the view is created and is not affected by subsequent changes to the underlying base tables. Aurora MySQL Views have the following restrictions:

- A view cannot reference system variables or user-defined variables.
- When used within a stored procedure or function, the SELECT statement cannot reference parameters or local variables.
- A view cannot reference prepared statement parameters.
- All objects referenced by a view must exist when the view is created. If an underlying table or view is later dropped, invoking the view results in an error.
- Views cannot reference TEMPORARY tables.
- TEMPORARY views are not supported.

- Views do not support triggers.
- Aliases are limited to a maximum length of 64 characters (not the typical 256 maximum alias length).

Aurora MySQL provides additional properties not available in Oracle:

- The ALGORITHM clause is a fixed hint that affects the way the MySQL query processor handles the view physical evaluation operator. The MERGE algorithm uses a dynamic approach where the definition of the view is merged to the outer query. The TEMPTABLE algorithm materializes the view data internally. For more information, see <https://dev.mysql.com/doc/refman/5.7/en/view-algorithms.html>
- The DEFINER and SQL SECURITY clauses can be used to specify a specific security context for checking view permissions at run time.

Similar to Oracle, Aurora MySQL supports updatable views and the ANSI standard CHECK OPTION to limit inserts and updates to rows referenced by the view.

The LOCAL and CASCADED keywords are used to determine the scope of violation checks. When using the LOCAL keyword, the CHECK OPTION is evaluated only for the view being created. CASCADED causes evaluation of referenced views. The default is CASCADED.

In general, only views having a one-to-one relationship between the source rows and the exposed rows are updatable. Adding the following constructs prevents modification of data:

- Aggregate functions
- DISTINCT
- GROUP BY
- HAVING
- UNION or UNION ALL
- Subquery in the select list
- Certain joins
- Reference to a non-updatable view
- Subquery in the WHERE clause that refers to a table in the FROM clause
- ALGORITHM = TEMPTABLE
- Multiple references to any column of a base table

A view must have unique column names. Column aliases are derived from the base tables or explicitly specified in the SELECT statement of column definition list. ORDER BY is permitted in Aurora MySQL, but ignored if the outer query has an ORDER BY clause.

A view in Aurora MySQL can invoke functions, which in turn may introduce a change to the database.

Aurora MySQL assesses data access privileges as follows:

- The user creating a view must have all required privileges to use the top-level objects referenced by the view. For example, for a view referencing table columns, the user must have privilege for each column in the select list of the view definition.
- If the view definition references a stored function, only the privileges needed to invoke the function are checked. The privileges required at run time can be checked only at run time because different invocations may use different execution paths within the function code.
- The user referencing a view must have the appropriate SELECT, INSERT, UPDATE, or DELETE privileges (as with a normal table).

- When a view is referenced, privileges for all objects accessed by the view are evaluated using the privileges for the view DEFINER account, or the invoker, depending on whether SQL SECURITY is set to DEFINER or INVOKER.
- When a view invocation triggers the execution of a stored function, privileges are checked for statements executed within the function based on the function's SQL SECURITY setting. For functions where the security is set to DEFINER, the function executes with the privileges of the DEFINER account. For functions where it is set to INVOKER, the function executes with the privileges determined by the view's SQL SECURITY setting as described above.

Syntax

```
CREATE [OR REPLACE]
  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  [DEFINER = { <User> | CURRENT_USER }]
  [SQL SECURITY { DEFINER | INVOKER }]
  VIEW <View Name> [(<Column List>)]
  AS <SELECT Statement>
  [WITH [CASCADED | LOCAL] CHECK OPTION];
```

Examples

Create and populate an Invoices table.

```
CREATE TABLE Invoices (
  InvoiceID      INT NOT NULL PRIMARY KEY,
  Customer      VARCHAR(20) NOT NULL,
  TotalAmount   DECIMAL(9,2) NOT NULL);

INSERT INTO Invoices (InvoiceID, Customer, TotalAmount)
VALUES (1, 'John', 1400.23), (2, 'Jeff', 245.00), (3, 'James', 677.22);
```

Create the TotalSales view.

```
CREATE VIEW TotalSales
AS
SELECT Customer, SUM(TotalAmount) AS CustomerTotalAmount
GROUP BY Customer;
```

Invoke the view.



```
SELECT * FROM TotalSales
ORDER BY CustomerTotalAmount DESC;
```

Customer	CustomerTotalAmount
John	1400.23
James	677.22
Jeff	245.00

For more information, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-view.html>
- <https://dev.mysql.com/doc/refman/5.7/en/view-restrictions.html>
- <https://dev.mysql.com/doc/refman/5.7/en/view-updatability.html>

Oracle XML DB vs. MySQL XML

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		XML	Different paradigm and syntax will require application/drivers rewrite.

Oracle Usage

Oracle XML DB is a set of Oracle Database technologies providing XML capabilities for database administrators and developers. It provides native XML support and other features including the native XMLType and XMLIndex.

XMLType represents an XML document in the database that is accessible from SQL. It supports standards such as XML Schema, XPath, XQuery, XSLT, and DOM.

XMLIndex supports all forms of XML data from highly structured to completely unstructured.

XML data can be schema-based or non-schema-based. Schema-based XML adheres to an XSD Schema Definition and must be validated. Non-schema-based XML data does not require validation.

According to Oracle documentation, the aspects you should consider when using XML are:

- The ways you intend to store your XML data.
- The structure of your XML data.
- The languages used to implement your application.
- The ways you intend to process your XML data.

The most common features are:

- **Storage Model:** Binary XML
- **Indexing:** XML search index, XMLIndex with structured component
- **Database language:** SQL, with SQL/XML functions
- **XML languages:** XQuery and XSLT

Storage Model – Binary XML

Also called “post-parse persistence”, it is the default storage model for Oracle XML DB. It is a post-parse, binary format designed specifically for XML data.

Binary XML is XML schema-aware and the storage is very flexible. You can use it for XML schema-based documents or for documents that are not based on an XML schema. You can use it with an XML schema that allows for high data variability or evolves considerably or unexpectedly. This storage model also provides efficient partial updating and streaming query evaluation.

The other storage option is “Object-relational storage” and is more efficient when using XML as structured data with a minimum amount of changes and different queries. For more information, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/adxdb/xml-db-developers-guide.pdf>.

Indexing – XML Search Index, XMLIndex with Structured Component

XML Search Index provides full-text search over XML data. Oracle recommends storing XMLType data as Binary XML and using XQuery Full Text (XQFT). If you are not using Binary storage and your data is structured XML, you can use the Oracle Text indexes, use the regular string functions such as “contains”, or use XPath “ora:-contains”. If you want to use predicates such as “XMLExists” in WHERE clauses, you must create an XML search index.

Examples

The following example creates an SQL directory object, which is a logical name in the database for a physical directory on the host computer. This directory contains XML files. The example inserts XML content from the purOrder.xml file into the orders table.

Create an XMLType table.

```
CREATE TABLE orders OF XMLType;
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file;
INSERT INTO orders VALUES
(XMLType(BFILENAME('XMLDIR','purOrder.xml'),NLS_CHARSET_ID('AL32UTF8')));
```

Create table with an XMLType column.

```
CREATE TABLE xwarehouses (warehouse_id NUMBER, warehouse_spec XMLTYPE);
```

Create an XMLType View.

```
CREATE VIEW warehouse_view AS
SELECT VALUE(p) AS warehouse_xml FROM xwarehouses p;
```

Insert data into an XMLType column.

```
INSERT INTO xwarehouses
VALUES(100, '<?xml version="1.0"?>
<PO pono="1">
<PNAME>Po_1</PNAME>
<CUSTNAME>John</CUSTNAME>
<SHIPADDR>
<STREET>1033, Main Street</STREET>
<CITY>Sunnyvalue</CITY>
<STATE>CA</STATE>
</SHIPADDR></PO>')
```

Create an XML Search Index and query it with XQuery:

1. Ensure the user has the appropriate permissions on the Oracle Text schema.
2. Create Oracle Text section and preference.
3. Create the XML Search Index (regular index associated with the objects).

```
BEGIN
CTX_DDL.create_section_group('secgroup', 'PATH_SECTION_GROUP');
CTX_DDL.set_sec_grp_attr('secgroup', 'XML_ENABLE', 'T');
```

```

CTX_DDL.create_preference('pref', 'BASIC_STORAGE');
CTX_DDL.set_attribute('pref','D_TABLE_CLAUSE', 'TABLESPACE ts_name LOB(DOC) STORE AS
SECUREFILE(TABLESPACE ts_name COMPRESS MEDIUM CACHE)');
CTX_DDL.set_attribute('pref','I_TABLE_CLAUSE','TABLESPACE ts_name LOB(TOKEN_INFO)
STORE AS SECUREFILE(TABLESPACE ts_name NOCOMPRESS CACHE)');
END;
/

CREATE INDEX po_ctx_idx ON po_binxml(OBJECT_VALUE)
INDEXTYPE IS CTXSYS.CONTEXT
PARAMETERS('storage pref section group secgroup');

```

Query using the above index in XQuery. XQuery is a W3C standard for generating, querying, and updating XML. It is the natural query language for XML.

Search in the PATH `"/PurchaseOrder/LineItems/LineItem/Description"` for values containing `"Big"` and `"Street"` and the return their `"Title"` tag (only in the select).

```

SELECT XMLQuery('for $i in /PurchaseOrder/LineItems/LineItem/Description where $i[.
contains text "Big" ftand "Street"] return <Title>{$i}</Title>' PASSING OBJECT_VALUE
RETURNING CONTENT)
FROM po_binxml
WHERE XMLElementExists('/PurchaseOrder/LineItems/LineItem/Description [. contains text "Big"
ftand "Street"]')

```

XMLIndex with Structured Component is used for queries that project fixed structured islands of XML content, even if the surrounding data is relatively unstructured. A structured XMLIndex component organizes such islands in a relational format. You must define the parts of XML data that you search in queries (applies to both XML schema-based and non-schema-based data).

Create an XMLIndex with structured component:

1. Create the base XMLIndex on po_binxml table. OBJECT_VALUE is the XML data stored in the table. All definitions of XML types and Objects are from the XDB schema in the database.
2. Use DBMS_XMLINDEX.registerParameter to add another structure to the index.
3. Create tables (po_idx_tab and po_index_lineitem) to store index data as structured data. The root of the PATH appears next to each table name in the XML data (/PurchaseOrder and /LineItem). Each column is another PATH in this root. Note that in the first table (po_idx_tab), the last column is XMLType. It takes everything under this PATH and saves it as an XML datatype.
4. Add the group structure to the index.

```

CREATE INDEX po_xmlindex_ix ON po_binxml (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex PARAMETERS ('PATH TABLE path_tab');
BEGIN
DBMS_XMLINDEX.registerParameter(
'myparam',
'ADD_GROUP GROUP po_item
XMLTable po_idx_tab '/PurchaseOrder'
COLUMNS reference VARCHAR2(30) PATH 'Reference',
requestor VARCHAR2(30) PATH 'Requestor',
username VARCHAR2(30) PATH 'User',
lineitem XMLType PATH 'LineItems/LineItem' VIRTUAL
XMLTable po_index_lineitem '/LineItem' PASSING lineitem
COLUMNS itemno BINARY_DOUBLE PATH '@ItemNumber',
description VARCHAR2(256) PATH 'Description',

```

```
partno      VARCHAR2(14)  PATH  'Part/@Id',
quantity   BINARY_DOUBLE PATH  'Part/@Quantity',
unitprice   BINARY_DOUBLE PATH  'Part/@UnitPrice');
END;
/

ALTER INDEX po_xmlindex_ix PARAMETERS ('PARAM myparam');
```

For more information, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/adxdb/indexes-for-XMLType-data.html#GUID-FE16BE28-035C-49EE-9683-6ED37876C3FC>

SQL/XML Functions

There are two main SQL/XML groups: SQL/XML publishing functions and SQL/XML query and update functions

SQL/XML Publishing Functions

SQL/XML publishing functions are SQL results generated from XML data (also called “SQL/XML generation functions”).

XMLQuery: Used in SELECT clauses to return the result as XMLType data (See the previous example for creating an XML search index).

XMLTable: Used in FROM clauses to get results using XQuery, and then insert the results into a virtual table (you can insert into existing database table). For example:

```
SELECT po.reference, li.*
FROM po_binaryxml p,
XMLTable('/PurchaseOrder' PASSING p.OBJECT_VALUE
COLUMNS
reference VARCHAR2(30) PATH 'Reference',
lineitem XMLType      PATH 'LineItems/LineItem') po,
XMLTable('/LineItem' PASSING po.lineitem
COLUMNS
itemno      NUMBER(38)      PATH '@ItemNumber',
description VARCHAR2(256) PATH 'Description',
partno      VARCHAR2(14)   PATH 'Part/@Id',
quantity    NUMBER(12, 2)  PATH 'Part/@Quantity',
unitprice   NUMBER(8, 4)   PATH 'Part/@UnitPrice') li;
```

XMLExists: Used in WHERE clauses to check if an XQuery expression returns a non-empty query sequence. If it does, it returns TRUE. Otherwise, it returns FALSE. In the following example, the query searches the purchaseorder table for PurchaseOrders where the “SpecialInstructions” tag is set to “Expedite”.

```
SELECT OBJECT_VALUE FROM purchaseorder
WHERE XMLExists('/PurchaseOrder[SpecialInstructions="Expedite"]'
PASSING OBJECT_VALUE);
```

XMLCast: Used in SELECT clauses to convert scalar values returned from XQuery to NUMBER, VARCHAR2, CHAR, CLOB, BLOB, REF, or XMLType. For example, after finding the objects that have SpecialInstructions set to Expedite,

XMLCast returns the “Reference” in each item as VARCHAR2(100).

```
SELECT XMLCast(XMLQuery('/PurchaseOrder/Reference' PASSING OBJECT_VALUE
RETURNING CONTENT) AS VARCHAR2(100)) "REFERENCE"
FROM purchaseorder
WHERE XMLEExists('/PurchaseOrder[SpecialInstructions="Expedite"]'
PASSING OBJECT_VALUE);
```

For more information, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/XMLELEMENT.html#GUID-DEA75423-00EA-4034-A246-4A774ADC988E>

SQL/XML Query and Update Functions

SQL/XML query and update functions are used to query and update XML content as part of regular SQL operations.

XMLQuery: See the example above.

UPDATE: In the following example, after finding the relevant item with XMLEExists in the set clause, the command sets the OBJECT_VALUE to a new file ('NEW-DAUSTIN-20021009123335811PDT.xml') located in the directory ('XMLDIR').

```
UPDATE purchaseorder po
SET po.OBJECT_VALUE = XMLType(bfilename('XMLDIR', 'NEW-DAUSTIN-
20021009123335811PDT.xml'), nls_charset_id('AL32UTF8'))
WHERE XMLEExists('$p/PurchaseOrder[Reference="DAUSTIN-20021009123335811PDT"]' PASSING
po.OBJECT_VALUE AS "p");
```

SQL and PL/SQL

SQL and PL/SQL is addressed in the SQL and PL/SQL sections of this document.

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/XMLQUERY.html#GUID-9E8D3220-2CF5-4C63-BDC2-0526D57B9CDB>

MySQL Usage

Aurora MySQL support for unstructured data is the opposite of Oracle. There is minimal support for XML, but a native JSON data type and more than 25 dedicated JSON functions.

XML Support

Aurora MySQL supports two XML functions: ExtractValue and UpdateXML.

ExtractValue accepts an XML document, or fragment, and an XPATH expression. The function returns the character data of the child (or element) matched by the XPATH expression. If there is more than one match, the function returns the content of child nodes as a space delimited character string. ExtractValue returns only CDATA (it does not return tags) and does not return sub-tags contained within a matching tag or its content.

For example:

```
SELECT ExtractValue ('<Root><Person>John</Person><Person>Jim</Person></Root>',
'/Root/Person');
```

Results: John Jim

UpdateXML is used to replace an XML fragment with another fragment using XPATH expressions similar to ExtractValue. If a match is found, it returns the new, updated XML. If there are no matches, or multiple matches, the original XML is returned.

For example:

```
SELECT UpdateXML ('<Root><Person>John</Person><Person>Jim</Person></Root>', '/Root',
'<Person>Jack</Person>')
```

Results: <Person>Jack</Person>

Note: Aurora MySQL does not support MySQL LOAD XML syntax . For more information about loading data into Aurora MySQL directly from Amazon S3, see [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket](#)

Summary


The following table identifies similarities, differences, and key migration considerations.

Feature	Oracle	Aurora MySQL
XML functions	XMLQuery, XPath, XMLTable, XMExists, and XMLCast	ExtractValue and UpdateXML.
Create table with XML	CREATE TABLE test OF XMLType; or CREATE TABLE test (doc XMLType);	Not supported.
Insert data into xml column	INSERT INTO test VALUES ('<?xml version="1.0"?> <PO pono="1"> <PNAME>Po_1</PNAME><CUSTNAME>John</CUSTNAME><SHIPADDR> <STREET>1033, Main Street</STREET> <CITY>Sunnyvalue</CITY><STATE>CA</STATE> </SHIPADDR></PO>')	XML data can be loaded into regular tables from S3. For more information see the AWS Documentation
Create Index	CREATE INDEX test_idx ON test (OBJECT_VALUE) INDEXTYPE IS XDB.XMLIndex PARAMETERS ('PATH TABLE path_tab'); BEGIN DBMS_XMLINDEX.registerParameter('myparam', 'ADD_GROUP GROUP a_item XMLTable test_idx_tab "/Path" COLUMNS tag VARCHAR2(30) PATH "tag"); END; / ALTER INDEX test_idx PARAMETERS ('PARAM myparam');	Requires adding always generated (computed and persisted) columns with JSON expressions and indexing them explicitly. The optimizer can make use of JSON expressions only.

Feature	Oracle	Aurora MySQL
Create Full-text Index	/*After preference and section created in Oracle Text*/ CREATE INDEX test_idx ON test (OBJECT_VALUE) INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS('storage pref section group sec-group');	N/A
Query using XQuery	SELECT XMLQuery('for \$i in /PurchaseOrder-/LineItems/LineItem/Description where \$i. contains text "Big" return <Title>{\$i}</Title>' PASSING OBJECT_VALUE RETURNING CONTENT) FROM xml_tbl;	N/A
Query using XPath	select sys.XMLType.extract (doc,'/student/firstname/text()') firstname from test;	Since there is no XML data type, doc uses VARCHAR to store the XML content. select ExtractValue (doc,'//student//firstname') firstname from test;
Function to check if tag exists	SELECT XMLCast(XMLQuery('/PurchaseOrder-/Reference' PASSING OBJECT_VALUE RETURNING CONTENT) AS VARCHAR2 (100)) "REFERENCE" FROM purchaseorder	N/A
Function to cast and return another data type(string)	WHERE XMLEExists('/PurchaseOrder[SpecialInstructions="Expedite"]' PASSING OBJECT_VALUE);	
Validate schema using XSD	Supported	Not supported.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/xml-functions.html>

Oracle Table Compression vs. MySQL InnoDB Table Compression

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Syntax and option differences, similar functionality Compress a partition are not supported

Oracle Usage

Oracle table compression reduces the size of data. It saves disk space, reduces memory usage, and speeds up query execution during reads. However, the cost is increased CPU overhead for data loading and DML.

Table compression is completely transparent to applications. It is most commonly used for OLAP systems where there are significantly more read operations, but it can also be used in OLTP systems.

Tables can be compressed when they are created using the COMPRESS clause. Existing tables can be compressed using the COMPRESS clause with an ALTER TABLE statement.

You can enable compression for ALL OPERATIONS on the table or for DIRECT_LOAD OPERATIONS only. When compression is enabled for all operations, compression occurs during all DML statements and when data is inserted with a bulk (direct-path) insert operation.

The compression clause provides four options:

- **NOCOMPRESS:** Do not use compression (default).
- **COMPRESS:** Compression is enabled on the table or partition during direct-path inserts only.
- **COMPRESS FOR DIRECT_LOAD OPERATIONS:** Same as COMPRESS.
- **COMPRESS FOR ALL OPERATIONS:** Enables compression for all operations including DML statements (mostly used for OLTP systems).

Examples

View the compression status of tables.

```
SELECT OWNER, TABLE_NAME, COMPRESSION, COMPRESS_FOR FROM dba_tables;
```

Create a compressed table.

```
CREATE TABLE comp_tbl
(id          NUMBER NOT NULL,
 created_date DATE NOT NULL)
COMPRESS FOR ALL OPERATIONS;
```

Create partitioned table with a compressed partition.

```
CREATE TABLE comp_part_tbl
(id          NUMBER NOT NULL,
 created_date DATE NOT NULL)
```

```

PARTITION BY RANGE (created_date) (
PARTITION comp_part_tbl_q1 VALUES LESS THAN (TO_DATE('01/01/2018', 'DD/MM/YYYY'))
COMPRESS,
PARTITION comp_part_tbl_q2 VALUES LESS THAN (TO_DATE('01/04/2018', 'DD/MM/YYYY'))
COMPRESS FOR DIRECT_LOAD OPERATIONS,
PARTITION comp_part_tbl_q3 VALUES LESS THAN (TO_DATE('01/07/2018', 'DD/MM/YYYY'))
COMPRESS FOR ALL OPERATIONS,
PARTITION comp_part_tbl_q4 VALUES LESS THAN (MAXVALUE) NOCOMPRESS);

```

For more details, see https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_COMPRESSION.htm-#GUID-9F37CAD6-C72C-407C-AFEE-CB5FD1129627

MySQL Usage

As with Oracle, MySQL provides support for compressing table data. Compression results in smaller database size, reduced I/O, and improved throughput at the cost of CPU overhead.

Tables created with ROW_FORMAT=COMPRESSED have a smaller page size on disk than the configured innodb_page_size value. Smaller pages require less I/O operations to read from, and write to, disk.

The compressed page size can be specified when tables are created with KEY_BLOCK_SIZE. Compressed tables must be placed in a file-per-table tablespace or general tablespace. The system tablespace cannot store compressed tables.

Compression works best on tables with a reasonable number of character string columns and where the data is used mostly for read operations. There is no way to predict whether or not compression will provide a benefit in a specific scenario. Always test your workload and data set on a representative configuration.

Unlike Oracle, MySQL does not support DIRECT LOAD or ALL OPERATIONS.

Examples

View table and index size.

```

SELECT
    table_name,
    round(((data_length + index_length) / 1024 / 1024), 2)
FROM information_schema.TABLES
WHERE table_schema = 'db_name'
AND table_name = 'table_name';

```

Create a compressed table with a 4k block size.

```

CREATE TABLE tbl_to_comp key_block_size=4 row_format=compressed;

```

Modify an existing table to be compressed with a 4k block size.

```

ALTER TABLE tbl_to_comp key_block_size=4 row_format=compressed;

```



Note: Compressed page size should be the maximum row size for an InnoDB table. KEY_BLOCK_SIZE=8 is a common choice.

Summary

Description	Oracle	MySQL
Create a compressed table	CREATE TABLE comp_tbl (col1 number) COMPRESS FOR ALL OPERATIONS;	CREATE TABLE tbl_to_comp key_ block_size=4 row_form- at=compressed;
Check which tables are compressed	SELECT table_name, COMPRESSION, COMPRESS_FOR FROM dba_tables;	SELECT * FROM information_schem- a.TABLES where CREATE_ OPTIONS like '%COMPRESSED%';
Compress a partition	ALTER TABLE sales MOVE PARTITION sales_ q1_2017 TABLESPACE ts_2017 COMPRESS;	Not supported

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/innodb-table-compression.html>

Oracle LogMiner vs. MySQL Logs and the mysql-binlog Utility

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL does not support LogMiner, workaround available

Oracle Usage

Oracle's LogMiner is a tool for querying the database Redo Logs and the Archived Redo Logs using a SQL interface. Using LogMiner, you can analyze the content of database “transaction logs” (online and archived redo logs) and gain historical insights on past database activity such as data modification by individual DML statements.

Examples

The following examples demonstrates how to use Log Miner to view DML statements executed on the employees table:

Find the current redo log file.

```
SELECT V$LOG.STATUS, MEMBER
FROM V$LOG, V$LOGFILE
WHERE V$LOG.GROUP# = V$LOGFILE.GROUP#
AND V$LOG.STATUS = 'CURRENT';
```

STATUS	MEMBER
-----	-----
CURRENT	/u01/app/oracle/oradata/orcl/redo02.log

Use the DBMS_LOGMNR.ADD_LOGFILE procedure. Pass the file path as a parameter to the Log Miner API.

```
BEGIN
DBMS_LOGMNR.ADD_LOGFILE('/u01/app/oracle/oradata/orcl/redo02.log');
END;
/

PL/SQL procedure successfully completed.
```

Start Log Miner using the DBMS_LOGMNR.START_LOGMNR procedure.

```
BEGIN
DBMS_LOGMNR.START_LOGMNR(options=>
dbms_logmnr.dict_from_online_catalog);
END;
/

PL/SQL procedure successfully completed.
```

Run a DML statement.

```
UPDATE HR.EMPLOYEES SET SALARY=SALARY+1000 WHERE EMPLOYEE_ID=116;
COMMIT;
```

Query the V\$LOGMNR_CONTENTS table to view the DML commands captured by the LogMiner.

```
SELECT TO_CHAR(TIMESTAMP, 'mm/dd/yy hh24:mi:ss') TIMESTAMP,
SEG_NAME, OPERATION, SQL_REDO, SQL_UNDO
FROM V$LOGMNR_CONTENTS
WHERE TABLE_NAME = 'EMPLOYEES'
AND OPERATION = 'UPDATE';
```

TIMESTAMP	SEG_NAME	OPERATION
10/09/17 06:43:44	EMPLOYEES	UPDATE

```
SQL_REDO                                SQL_UNDO
-----                                -
```

update "HR"."EMPLOYEES" set "SALARY" = '3900' where "SALARY" = '2900' and ROWID = 'AAAViUAAEAAAABVvAAQ';	update "HR"."EMPLOYEES" set "SALARY" = '2900' where "SALARY" = '3900' and ROWID = 'AAAViUAAEAAAABVvAAQ';
---	---

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sutil/oracle-logminer-utility.html#GUID-3417B738-374C-4EE3-B15C-3A66E01AE2B5>

MySQL Usage

The mysqlbinlog utility is the MySQL equivalent to Oracle's LogMiner, but LogMiner can be used for searching many types of information. This topic will cover all of the MySQL logs that are available so you can decide which log is best for your use-case.

Aurora MySQL generates four logs that can be viewed by database administrators:

- **Error Log:** Contains information about errors and server start/stop events.
- **General Query Log:** Contains a general record of mysqld operations (connect, disconnect, queries, etc.).
- **Slow Query Log:** Contains a log of "slow" SQL statements.
- **Bin Log:** When used, contain row and statement levels commands records.

The MySQL error log is generated by default. You can generate the slow query and general logs by setting parameters in the database parameter group. Amazon RDS rotates all MySQL log files.

You can monitor the MySQL logs directly through the Amazon RDS console, Amazon RDS API, AWS CLI, or AWS SDKs. You can also access MySQL logs by directing the logs to a database table in the main database and then querying that table. You can use the mysqlbinlog utility to download a binary log.

Downloading MySQL Binlog files

The binlog in MySQL is used for replication needs. MySQL uses it to replicate commands between master MySQL server to slave server. These logs can be read using the mysqlbinlog utility.

The mysqlbinlog utility is equivalent to Oracle's LogMiner and allows you to read the server's binary log (similar to the Oracle redo log). The server's binary log consists of files that describe modifications to database contents (events).

While these logs do not contain a lot of information, they can provide needed data for some use cases.

To download and read the binary log, check to see if the binlog is activated by typing this command:

```
SHOW BINARY LOGS;
```

Note: If the binlog is not activated, this command returns an error. If the binlog is activated, the binlog files list is displayed.

After querying the binlog files list you can select a file to download by using this command:

```
mysqlbinlog --read-from-remote-server --host=mysql-cluster1.cluster-crqdlsqlqnpny.us-east-1.rds.amazonaws.com --port=3306 --user naya --password mysql-bin-changelog.0098
```

For more details about this command : [MySQL Database Log Files](#)

Output example for binlog:

```
use `aws`/*!*/;
SET TIMESTAMP=1551125550/*!*/;
SET @@session.pseudo_thread_id=12/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=2097152/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_server=8/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
  last_committed=1          sequence_number=2          rbr_only=no          original_committed_
timestamp=0  immediate_commit_timestamp=0          transaction_length=0
# original_commit_timestamp=0 (1969-12-31 19:00:00.000000 Eastern Standard Time)
# immediate_commit_timestamp=0 (1969-12-31 19:00:00.000000 Eastern Standard Time)
/*!80001 SET @@session.original_commit_timestamp=0*//*!*/;
/*!80014 SET @@session.original_server_version=0*//*!*/;
/*!80014 SET @@session.immediate_server_version=0*//*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 434
#190225 15:12:50 server id 565151648  end_log_pos 513 CRC32 0x1188c639  Query
thread_id=12  exec_time=0          error_code=0
SET TIMESTAMP=1551125570/*!*/;
BEGIN
/*!*/;
# at 513
#190225 15:12:50 server id 565151648  end_log_pos 669 CRC32 0x051c3800  Query
thread_id=12  exec_time=0          error_code=0
SET TIMESTAMP=1551125570/*!*/;
/* ApplicationName=mysql */ insert into test values (1),(1),(1)
/*!*/;
# at 669
#190225 15:12:50 server id 565151648  end_log_pos 700 CRC32 0x72697ff4  Xid = 5467
```

```
COMMIT/*!*/;  
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;  
DELIMITER ;  
# End of log file
```

For additional details, see: <https://dev.mysql.com/doc/refman/5.7/en/mysqlbinlog.html>

Accessing MySQL Error Logs

The MySQL error log is written to the `mysql-error.log` file. You can view `mysql-error.log` by using the Amazon RDS console or by retrieving the log using the Amazon RDS API, Amazon RDS CLI, or AWS SDKs. `mysql-error.log` is flushed every 5 minutes and its contents are appended to `mysql-error-running.log`. The `mysql-error-running.log` file is then rotated every hour. The hourly files generated during the last 24 hours are retained. Each log file has the hour it was generated (in UTC) appended to its name. The log files also have a timestamp that helps you determine when the log entries were written.

MySQL writes to the error log only on startup, shutdown, and when it encounters errors. A database instance can go hours or days without new entries being written to the error log. If you see no recent entries, it's because the server did not encounter an error that would result in a log entry.

Accessing the MySQL Slow Query and General Logs

The MySQL slow query log and the general log can be written to a file or a database table by setting parameters in the database parameter group. You must set these parameters before you can view the slow query log or general log in the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs.

You can control MySQL logging by using the following parameters:

- **slow_query_log:** To create the slow query log, set to 1. The default is 0.
- **general_log:** To create the general log, set to 1. The default is 0.
- **long_query_time:** To prevent fast-running queries from being logged in the slow query log, specify a value for the shortest query execution time (in seconds) to be logged. The default is 10 seconds; the minimum is 0. If `log_output = FILE`, you can specify a floating point value with a resolution of microseconds. If `log_output = TABLE`, you must specify an integer value with a resolution of seconds. Only queries where the execution time exceeds the `long_query_time` value are logged. For example, setting `long_query_time` to 0.1 prevents a query that runs for less than 100 milliseconds from being logged.
- **log_queries_not_using_indexes:** To log all queries that do not use an index to the slow query log, set to 1. The default is 0. Queries that do not use an index are logged even if their execution time is less than the value of the `long_query_time` parameter.
- **log_output option:** You can specify one of the following options for the `log_output` parameter.
 - **TABLE (default):** Write general queries to the `mysql.general_log` table, and write slow queries to the `mysql.slow_log` table.
 - **FILE:** Write both general and slow query logs to the file system. Log files are rotated hourly.
 - **NONE:** Disable logging.

You can configure a MySQL instance to publish log data to a log group in Amazon CloudWatch Logs. CloudWatch Logs support real-time analysis of the log data, create alarms, and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage. For more information, see [MySQL Database Log Files](#)

Amazon RDS normally purges a binary log as soon as possible, but the binary log must still be available on the instance to be accessed by `mysqlbinlog`. To specify the number of hours for RDS to retain binary logs, use the `mysql.rds_set_configuration` stored procedure and specify a period with enough time for you to download the logs. After you set the retention period, monitor storage usage for the database instance to ensure the retained binary logs don't consume too much storage.

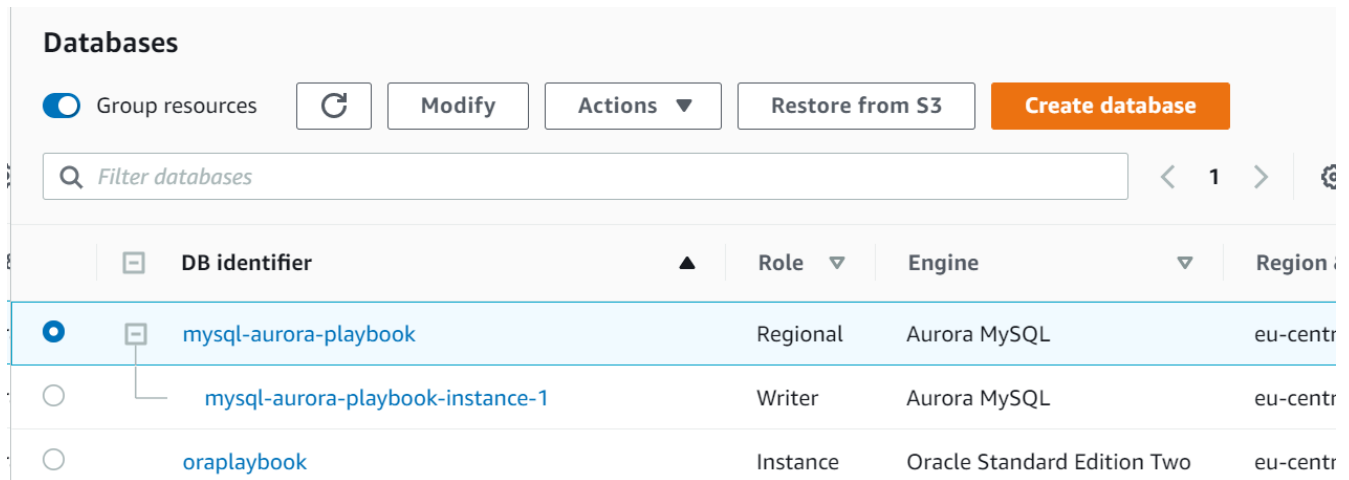
Examples

Determine the output location of the logs and if slow query and general logging are enabled.

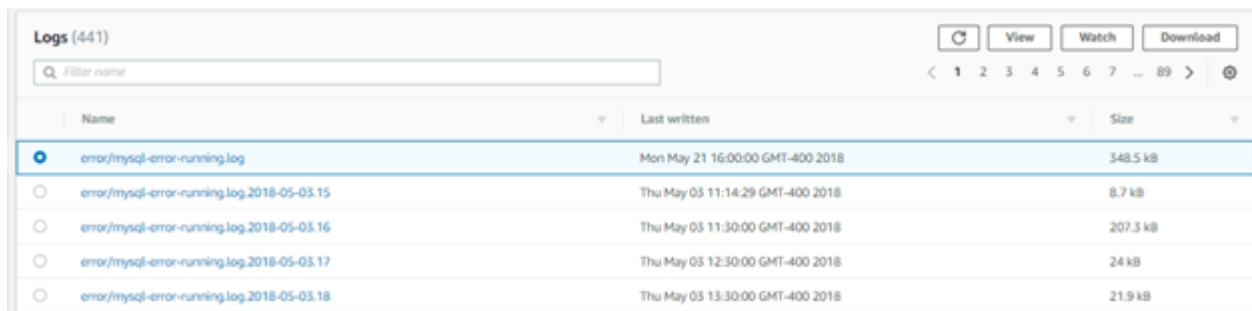
```
select @@GLOBAL.log_output, @@GLOBAL.slow_query_log, @@GLOBAL.general_log
```

View the logs using the console:

1. Navigate to the [RDS databases page](#).
2. Click the instance.



3. Scroll down to the **Logs** section.
4. Select a log to inspect or download.





Configure retention of the binary logs (in hours). In this example the binary log will be retained one day.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/binary-log.html>
- [MySQL Database Log Files](#)

Oracle SQL Result Cache vs. MySQL Query Cache

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	<p>Syntax and option differences, similar functionality</p> <p>Off the MySQL roadmap suggested not to be used</p>

Oracle Usage

The Oracle SQL Result Cache feature is related to the following caching categories:

- Global temporary tables
- Materialized Views
- PL/SQL collection
- The WHEN clause

The Result Cache reduces I/O operations by skipping the fetch step of execution plans and retrieving rows from the buffer cache. This feature is most useful for data warehouse scenarios where many rows must be scanned, but the result sets contain few rows. The rows are stored in the System Global Area (SGA) and are reused when the same SQL statements are executed in the current session or other sessions.

The `RESULT_CACHE_MODE` parameter controls caching and accepts the following values:

- **MANUAL**: SQL results are not cached for SQL statements (unless they use a hint to perform caching).
- **FORCE** : All results are cached for SQL statements (unless they use a hint to prevent caching).

Notes:

- In Oracle Real Application Cluster (RAC) environments, each instance has its own private result cache and can't be used by other instances.
- The query result cache is not compatible with scalar subquery caching.

Examples

Cache a query when `RESULT_CACHE_MODE` is set to `MANUAL`.

```
SELECT /*+ RESULT_CACHE */ count(*) FROM bigdata_smallres_tbl;
```

Disable caching when `RESULT_CACHE_MODE` is set to `FORCE` and a result cache is not needed .

```
SELECT /*+ NO_RESULT_CACHE */ count(*) FROM bigdata_smallres_tbl;
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/tgdba/tuning-result-cache.htm-#GUID-21CAA1E7-9E46-4442-9F3E-CE09EEF60D92>

MySQL Usage

RDS MySQL Only (not Aurora MySQL): The Query Cache is deprecated as of MySQL 5.7.20 and will be removed in MySQL 8.0. For more information, see [MySQL 8.0: Retiring Support for the Query Cache](#).

In accordance with the MySQL road map, it is recommended to not use the Query Cache.

Like the Oracle Result Cache, the MySQL Query Cache reduces I/O operations by skipping the fetch step of execution plans and retrieving rows from the buffer cache. It can be shared across multiple sessions.

Examples

Execute a select statement using the Query Cache.

```
SELECT SQL_CACHE count(*) FROM bigdata_smallres_tbl;
```


Execute a select statement without using the Query Cache.

```
SELECT SQL_NO_CACHE count(*) FROM bigdata_smallres_tbl;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/query-cache.html>

High Availability and Disaster Recovery (HADR)

Oracle Active Data Guard vs. MySQL Replicas

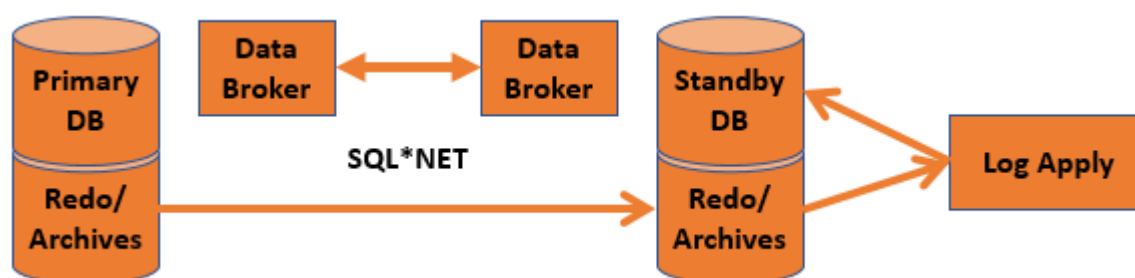
Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Distribute load/applications/users across multiple instances

Oracle Usage

Oracle Active Data Guard (ADG) is a synced database architecture with a primary and standby databases. The differences between Data Guard and Active Data Guard are:

- Active Data Guard allows you to open the database for read-only (e.g. reports) while it is still being kept in sync with your primary database.
- The normal Data Guard architecture requires you to pause the syncing process if you open the database for read-only.

The following diagram illustrates the ADG architecture :



- **Primary DB:** The main database open to read and write operations.
- **Redo/Archive** The redo files and archives that store the redo entries for recovery operations.
- **Data Broker:** The data guard broker service is responsible for all fail-over and syncing operations.
- **Standby DB:** The secondary database that allows read operations only. This database remains in recovery mode until it is shut down or becomes the primary (fail-over or switchover).
- **Log Apply** Runs all the redo log entries from the redo and archives files on the standby db.
- **Redo/Archive:** Contains the redo files and archives that are synced from the primary log and archive files.
- **Data Broker:** The Data Guard broker service is responsible for all fail-over and syncing operations.
- All components use SQL*NET protocol.

Special features:

- You can select "asynchronously" for best performance or "synchronously" for best data protection.
- You can temporarily convert a standby database to a snapshot database and allow read/write operations. When you are done running QA, testing, loads, or other operations, it can be switched back to standby.
- A sync gap can be specified between the primary and standby databases to account for human errors (for example creating 12 hours gap of sync).

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/sbydb/creating-oracle-data-guard-physical-standby.html#GUID-B511FB6E-E3E7-436D-94B5-071C37550170>

MySQL Usage

Aurora replicas can be used for scaling read operations and increasing availability like Oracle Active Data Guard, but with less configuration and administration. You can easily manage many replicas from the RDS console. Alternatively, you can use the AWS CLI for automation.

When creating Aurora MySQL instances, there are two replication options:

- **Multi-AZ (Availability Zone):** Create a replicating instance in a different region.
- **Instance Read Replicas:** Create a replicating instance in the same region.

In "Instance options" there are two options for this scenario:

- Create Aurora Replica
- Create Cross Region Read Replica

The main differences between the two options are:

- Cross Region creates a new reader cluster in a different region. Use Cross Region for a higher level of Higher Availability and to keep the data closer to the end users.
- Cross Region has more lag between the two instances.
- Additional charges apply for transferring the data between the two regions.

To view the most current lag value between the primary and replicas, query the `mysql.ro_replica_status` table and check the `Replica_lag_in_msec` column. This column value is provided to Amazon CloudWatch as the `ReplicaLag` metric. The values in the `mysql.ro_replica_status` are also provided in the `INFORMATION_SCHEMA.REPLICA_HOST_STATUS` table in your Aurora MySQL DB cluster.

DDL statements executed on the primary instance may interrupt database connections on the associated Aurora Replicas. If an Aurora Replica connection is actively using a database object such as a table, and that object is modified on the primary instance using a DDL statement, the Aurora Replica connection is interrupted.

Rebooting the primary instance of an Amazon Aurora database cluster also automatically reboots the Aurora Replicas for that database cluster.

Before creating a cross region replica, the `binlog_format` parameter must be enabled.

When using Multi-AZ, the primary database instance switches over automatically to the standby replica if any of the following conditions occur:

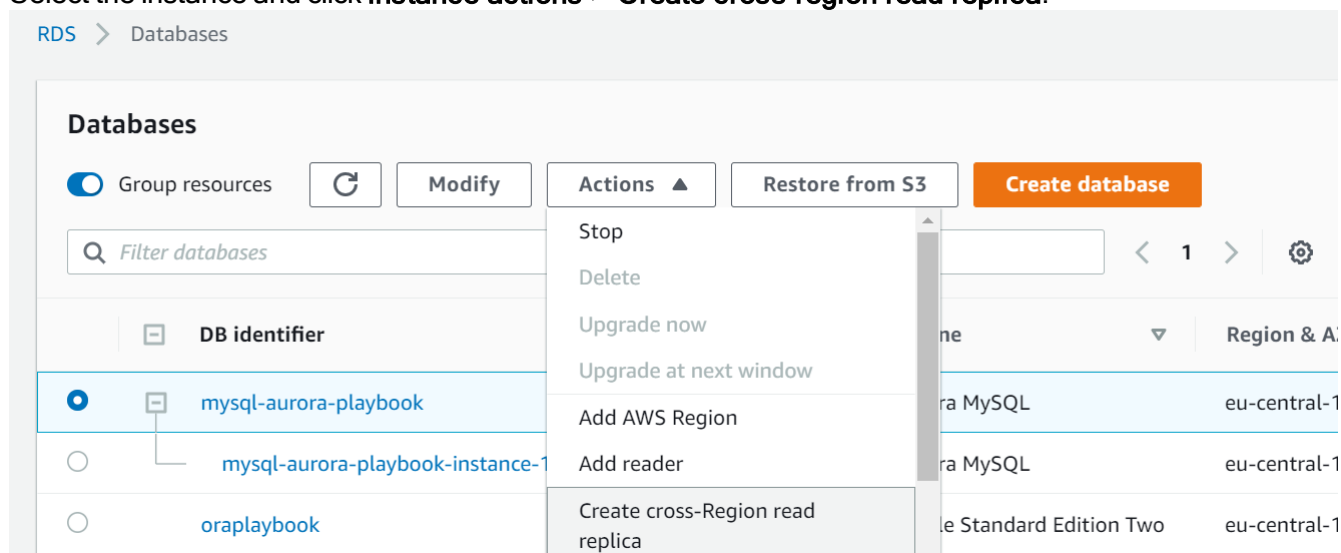
- The primary database instance fails.
- An Availability Zone outage.

- The database instance server type is changed.
- The operating system of the database instance is undergoing software patching.
- A manual failover of the database instance was initiated using reboot with failover.

Examples

The following walk-through demonstrates how to create a read-replica:

1. Navigate to the [RDS databases page](#).
2. Select the instance and click **Instance actions** > **Create cross region read replica**.



3. On the next page, enter all required details and click **Create**.

After the replica is created, you can execute read and write operations on the primary instance and read-only operations on the replica.

For more details, see:

- [Replication with Amazon Aurora MySQL](#)
- [Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions](#)
- [Creating an Amazon Aurora DB Cluster](#)

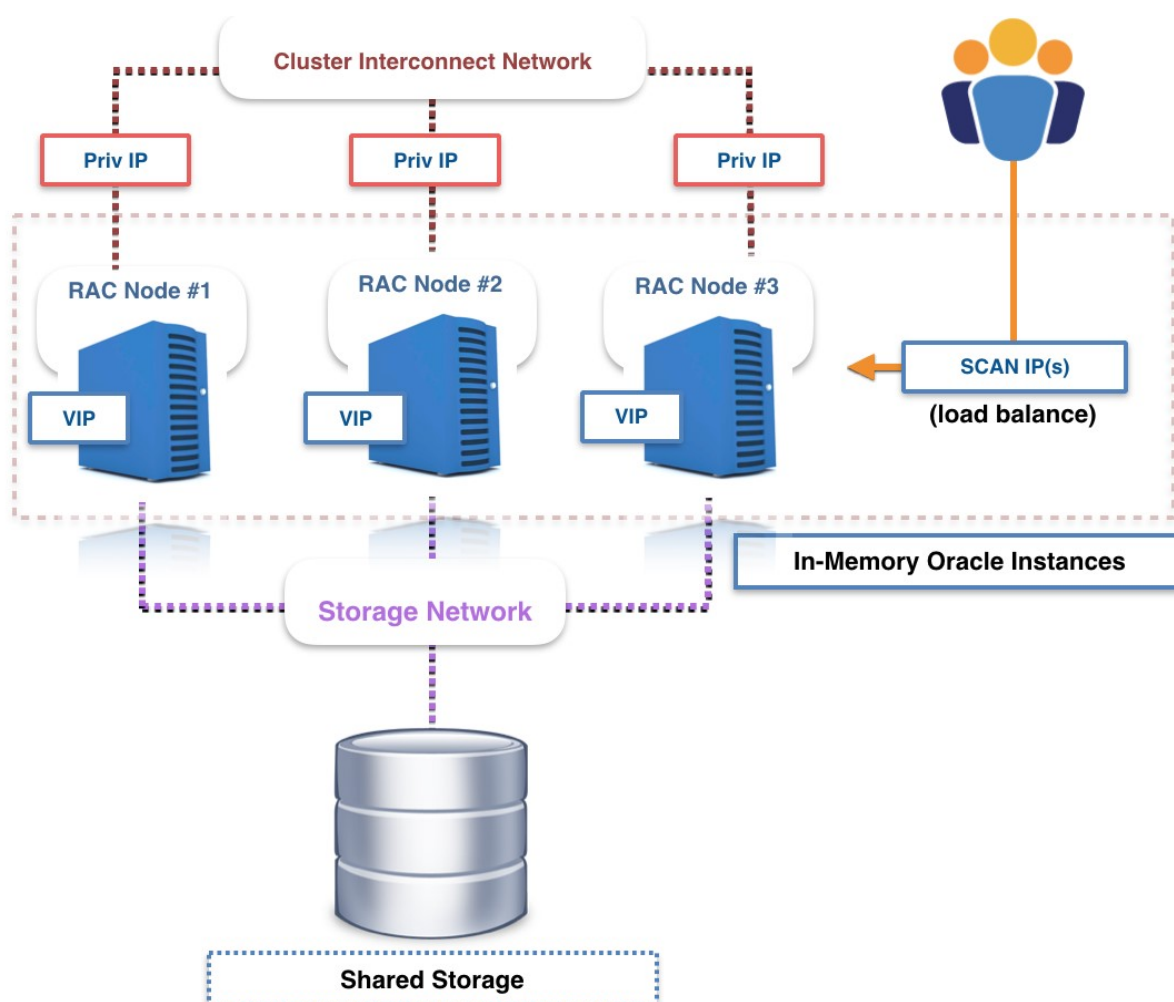
Oracle Real Application Clusters (RAC) vs. MySQL Aurora Architecture

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Distribute load/applications/ users across multiple instances

Oracle Usage

Oracle Real Application Clusters (RAC) is one of the most advanced and capable technologies providing highly available and scalable relational databases. It allows multiple Oracle instances to access a single database. Applications can access the database through the multiple instances in Active-Active mode.

The following diagram illustrates the Oracle RAC architecture:



Oracle RAC requires network configuration of SCAN IPs, VIP IPs, interconnect, and other items. As a best practice, all servers should run the same versions of Oracle software.

Because of the shared nature of the RAC cluster architecture—specifically, having all nodes write to a single set of database data files on disk—the following two special coordination mechanisms ensure Oracle database objects and data maintain ACID compliance:

- **GCS (Global Cache Services):** Tracks the location and status of the database data blocks and helps guarantee data integrity for global access across all cluster nodes.
- **GES (Global Enqueue Services):** Performs concurrency control across all cluster nodes including cache locks and transactions.

These services, which run as background processes on each cluster node, are essential for serializing access to shared data structures in an Oracle database.

Shared storage is another essential component in the Oracle RAC architecture. All cluster nodes read and write data to the same physical database files stored on a disk accessible by all nodes. Most customers rely on high-end storage hardware to provide the shared storage capabilities required for RAC.

In addition, Oracle provides its own software-based storage/disk management mechanism called Automatic Storage Management (ASM). ASM is implemented as a set of special background processes that run on all cluster nodes and allow for easy management of the database storage layer.

Performance and Scale-Out with Oracle RAC

You can add new nodes to an existing RAC cluster without downtime. Adding more nodes increases the level of high availability and enhances performance.

Although you can scale read performance easily by adding more cluster nodes, scaling write performance is more complicated. Technically, Oracle RAC can scale writes and reads together when adding new nodes to the cluster, but attempts from multiple sessions to modify rows that reside in the same physical Oracle block (the lowest level of logical I/O performed by the database) can cause write overhead for the requested block and impact write performance.

Concurrency is another reason why RAC implements a “smart mastering” mechanism that attempts to reduce write-concurrency overhead. The “smart mastering” mechanism enables the database to determine which service causes which rows to be read into the buffer cache and master the data blocks only on those nodes where the service is active. Scaling writes in RAC isn’t as straightforward as scaling reads.

With the limitations for pure write scale-out, many Oracle RAC customers choose to split their RAC clusters into multiple “services,” which are logical groupings of nodes in the same RAC cluster. By using services, you can use Oracle RAC to perform direct writes to specific cluster nodes. This is usually done in one of two ways:

- Splitting writes from different individual “modules” in the application (that is, groups of independent tables) to different nodes in the cluster. This approach is also known as “application partitioning” (not to be confused with database table partitions).
- In extremely non-optimized workloads with high concurrency, directing all writes to a single RAC node and load-balancing only the reads.

In summary, Oracle Real Application Clusters provides two major benefits:

- Multiple database nodes within a single RAC cluster provide increased high availability. No single point of failure exists from the database servers themselves. However, the shared storage requires storage-based high availability or disaster recovery solutions.
- Multiple cluster database nodes enable scaling-out query performance across multiple servers.

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/racad/index.html>

MySQL Usage

Aurora extends the “vanilla” versions of MySQL in two major ways:

- Adds enhancements to the MySQL database kernel itself to improve performance (concurrency, locking, multi-threading, etc.).
- Uses the capabilities of the AWS ecosystem for greater high availability, disaster recovery, and backup/recovery functionality.

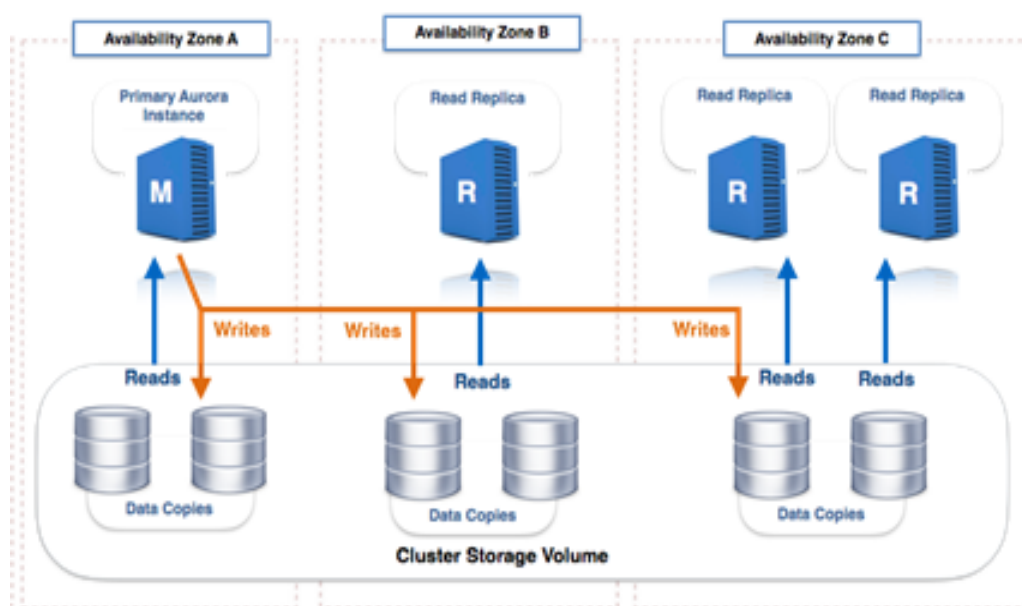
Comparing the Amazon Aurora architecture to Oracle RAC, there are major differences in how Amazon implements scalability and increased high availability. These differences are due mainly to the existing capabilities of MySQL and the strengths the AWS backend provides in terms of networking and storage.

Instead of having multiple read/write cluster nodes access a shared disk, an Aurora cluster has a single primary node (“master”) that is open for reads and writes and a set of replica nodes that are open for reads with automatic promotion to primary (“master”) in case of failures. While Oracle RAC uses a set of background processes to coordinate writes across all cluster nodes, the Amazon Aurora Master writes a constant redo stream to six storage nodes distributed across three Availability Zones within an AWS Region. The only writes that cross the network are redo log records (not pages).

Each Aurora cluster can have one or more instances serving different purposes:

- At any given time, a single instance functions as the primary (“master”) that handles both writes and reads from your applications.
- Up to 15 read replicas can be created in addition to the primary (“master”), which are used for two purposes:
 - **Performance and Read Scalability:** Replicas can be used as read-only nodes for queries and report workloads.
 - **High Availability:** Replicas can be used as failover nodes in the event the master fails. Each read replica can be located in one of the three Availability Zones hosting the Aurora cluster. A single Availability Zone can host more than one read replica.

The following diagram illustrates a high-level Aurora architecture with four cluster nodes: one primary (“master”) and three read replicas. The primary node is located in Availability Zone A, the first read replica in Availability Zone B, and the second and third read replicas in Availability Zone C.



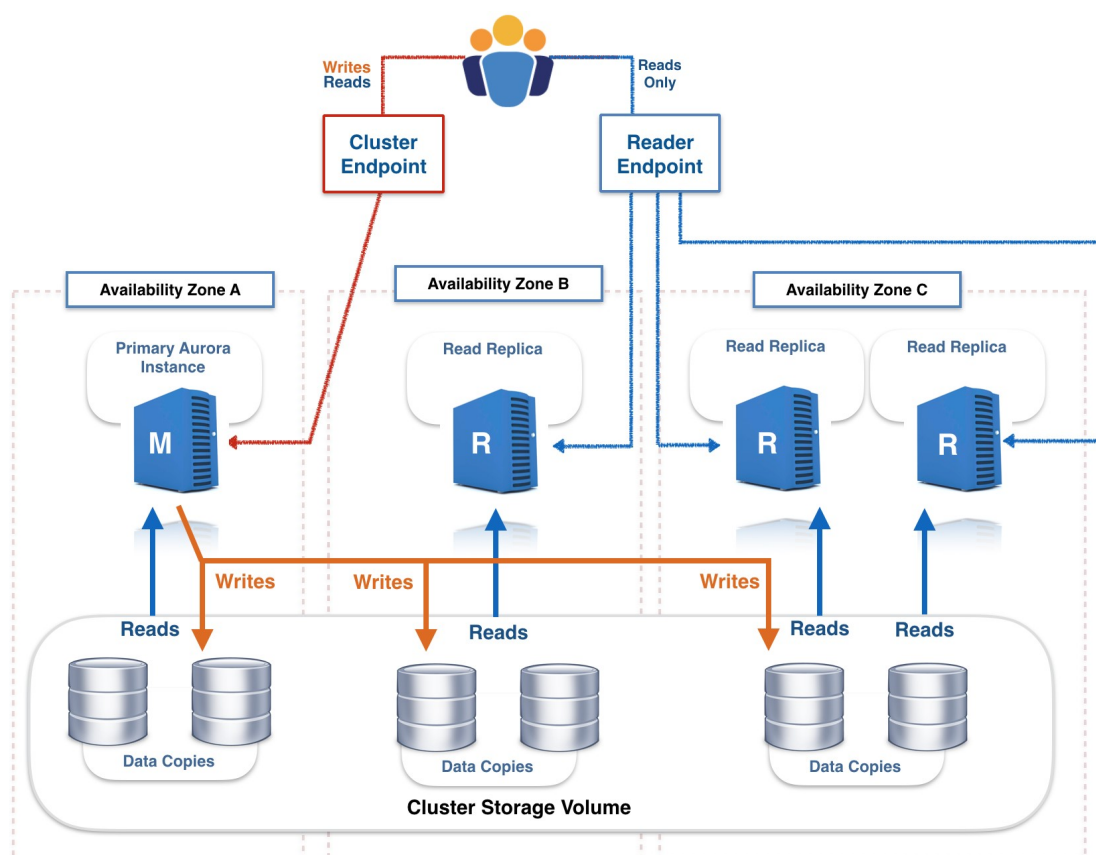
An Aurora Storage volume is made up of 10 GB segments of data with six copies spread across three Availability Zones. Each Amazon Aurora read replica shares the same underlying volume as the master instance. Updates made by the master are visible to all read replicas through a combination of reading from the shared Aurora storage volume and applying log updates in-memory when received from the primary instance after a master failure. Promotion of a read replica to master usually occurs in less than 30 seconds with no data loss.

For a write to be considered durable in Aurora, the primary instance (“master”) sends a redo stream to six storage nodes – two in each availability zone for the storage volume – and waits until four of the six nodes have responded. No database pages are ever written from the database tier to the storage tier. The Aurora Storage volume asynchronously applies redo records to generate database pages in the background or on demand. Aurora hides the underlying complexity.

High availability and Scale-Out in Aurora

Aurora provides two endpoints for cluster access. These endpoints provide both high availability capabilities and scale-out read processing for connecting applications

- **Cluster Endpoint:** Connects to the current primary instance for the Aurora cluster. You can perform both read and write operations using the cluster endpoint. If the current primary instance fails, Aurora automatically fails over to a new primary instance. During a failover, the database cluster continues to serve connection requests to the cluster endpoint from the new primary instance with minimal interruption of service.
- **Reader Endpoint:** Provides load-balancing capabilities (round-robin) across the replicas allowing applications to scale-out reads across the Aurora cluster. Using the Reader Endpoint provides better use of the resources available in the cluster. The reader endpoint also enhances high availability. If an AWS Availability Zone fails, the application’s use of the reader endpoint continues to send read traffic to the other replicas with minimal disruption.

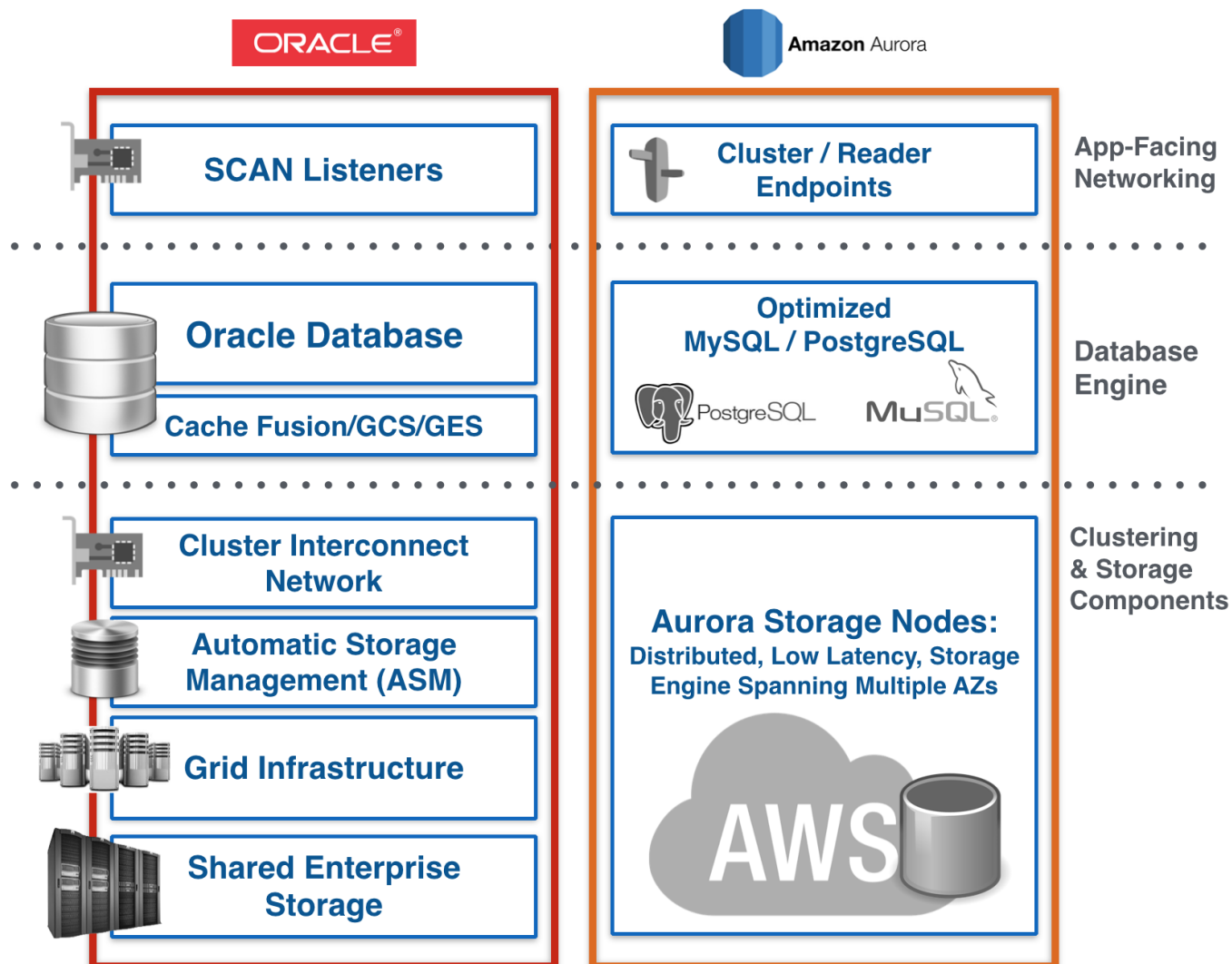


While Amazon Aurora focuses on the scale-out of reads and Oracle RAC can scale-out both reads and writes, most OLTP applications are usually not limited by write scalability. Many Oracle RAC customers use RAC first for high availability and second to scale-out their reads. You can write to any node in an Oracle RAC cluster, but this capability is often a functional benefit for the application versus a method for achieving unlimited scalability for writes.

Summary

- In Aurora MySQL, multiple cluster database nodes provide increased high availability. There is no single point of failure from the database servers. In addition, since an Aurora cluster can be distributed across three availability zones, there is a large benefit for high availability and durability of the database. These types of “stretch” database clusters are usually uncommon with other database architectures.
- AWS managed storage nodes also provide high availability for the storage tier. A zero-data loss architecture is employed in the event a master node fails and a replica node is promoted to the new master. This failover can usually be performed in under 30 seconds.
- Multiple cluster database nodes enable scaling-out query read performance across multiple servers.
- Greatly reduced operational overhead using a cloud solution and reduced total cost of ownership by using AWS and open source database engines.
- Automatic management of storage. No need to pre-provision storage for a database. Storage is automatically added as needed, and you only pay for one copy of your data.
- With Amazon Aurora, you can easily scale-out your reads (and scale-up your writes) which fits perfectly into the workload characteristics of many, if not most, OLTP applications. Scaling out reads usually provides the most tangible performance benefit.

When comparing Oracle RAC and Amazon Aurora side by side, you can see the architectural differences between the two database technologies. Both provide high availability and scalability, but with different architectures.



Overall, Amazon Aurora introduces a simplified solution that can function as an Oracle RAC alternative for many typical OLTP applications that need high performance writes, scalable reads, and very high availability with lower operational overhead.

Summary

Feature	Oracle RAC	Amazon Aurora
Storage	Usually enterprise-grade storage + ASM	Aurora Storage Nodes: Distributed, Low Latency, Storage Engine Spanning Multiple

Feature	Oracle RAC	Amazon Aurora
		AZs
Cluster type	Active/Active · All nodes open for R/W	Active/Active : Primary node open for R/W, Replica nodes open for reads
Cluster virtual IPs	R/W load balancing: SCAN IP	R/W: Cluster endpoint + Read load balancing: Reader endpoint
Internode coordination	Cache-fusion + GCS + GES	---
Internode private network	Interconnect	---
Transaction (write) TTR from node failure	Typically 0-30 seconds	Typically < 30 Seconds
Application (Read) TTR from node failure	Immediate	Immediate
Max number of cluster nodes	Theoretical maximum is 100, but smaller clusters (2 - 10 nodes) are far more common	15
Provides built-in read scaling	Yes	Yes
Provides built-in write scaling	Yes (Under certain scenarios, write performance can be limited and affect scale-out capabilities. For example, when multiple sessions attempt to modify rows contained in the same database block(s).)	No
Data loss in case of node failure	No data loss	No data loss
Replication latency	---	Milliseconds
Operational complexity	Requires database, IT, network, and storage expertise	Provided as a cloud-solution
Scale-up nodes	Difficult with physical hardware, usually requires to replace servers	Easy using the AWS UI/CLI
Scale-out cluster	Provision, deploy, and configure new servers, unless you pre-allocate a pool of idle servers to scale-out on	Easy using the AWS UI/CLI

For more details, see [Amazon Aurora as an Alternative to Oracle RAC](#)

Migrate to Aurora MySQL Serverless

Usage

Another great option can be Amazon Aurora MySQL using Serverless option, this option is currently available only with Aurora MySQL 5.6 compatible

Amazon Aurora Serverless is an on-demand, auto-scaling configuration for Amazon Aurora (MySQL-compatible edition), where the database will automatically start up, shut down, and scale capacity up or down based on your application's needs. It enables you to run your database in the cloud without managing any database instances. It's a simple, cost-effective option for infrequent, intermittent, or unpredictable workloads.

Manually managing database capacity can take up valuable time and can lead to inefficient use of database resources. With Aurora Serverless, you simply create a database endpoint, optionally specify the desired database capacity range, and connect your applications. You pay on a per-second basis for the database capacity you use when the database is active, and migrate between standard and serverless configurations with a few clicks in the Amazon RDS Management Console.

For some use cases, this option can be very easy to integrate and it has a big advantage over the Oracle RAC in terms of costs. This instance can be adjusted according to your work load and this is more relevant in terms of scale-out for performance.

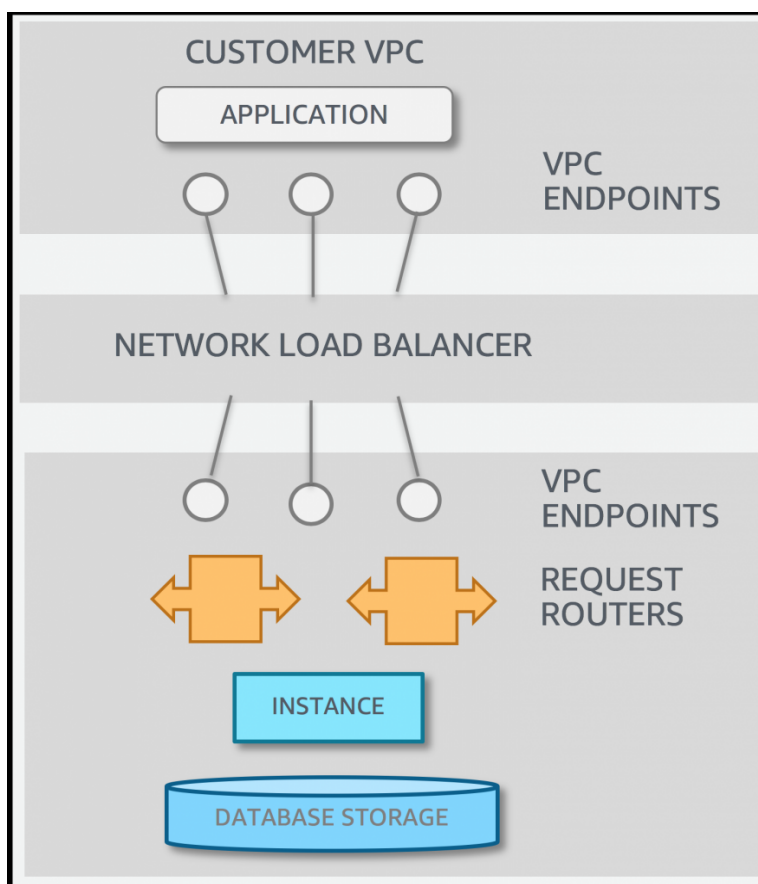
You can set the minimum and maximum capacity units required. By doing that, your MySQL serverless instance will scale in/out automatically according to the current workload.

These are the available capacity units:

1. CPU: 2, RAM: 4GB
2. CPU: 4, RAM: 8GB
3. CPU: 8, RAM: 16GB
4. CPU: 16, RAM: 32GB
5. CPU: 32, RAM: 64GB
6. CPU: 64, RAM: 122GB
7. CPU: 128, RAM: 244GB
8. CPU: 256, RAM: 488GB

How it works



- It creates an Aurora storage volume replicated across multiple AZs.
- It creates an endpoint in your VPC for the application to connect to.
- It configures a network load balancer (invisible to the customer) behind that endpoint.
- It configures multi-tenant request routers to route database traffic to the underlying instances.
- It provisions the initial minimum instance capacity.



This option can be easier than using Oracle RAC because with this option there is no need to add/remove servers from the cluster and there is no need to spend on unused hardware, it can scale out to even more than you thought you will need when the cluster was created.

For additional details, see: [Introduction Amazon Aurora Serverless](#)

Oracle Traffic Director vs. Amazon RDS Proxy for Amazon Aurora MySQL

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	Some features may be replaced by AWS RDS Proxy

Usage

Starting with Oracle 18c Oracle Connection Manager can be configured to run in Traffic Director mode. This mode introduces multiple features that help with High Availability, scalability, load balancing, zero downtime and security. Oracle Traffic Director is fast and reliable load-balancing solution. By enabling it for Oracle Connection Manager users can now get following features:

- Increased scalability through usage of transparent connection load-balancing
- Essential high availability feature of zero downtime that includes support for
 - planned database maintenance
 - pluggable database relocation
 - unplanned database outages for read-mostly workloads
- High availability of Connection Manager (CMAN) which avoids single point of failure
- Various security features, such as
 - database proxy
 - firewall
 - tenant isolation in multi-tenant environment
 - DDOS protection
 - Database traffic secure tunneling

Further reading: <https://docs.oracle.com/en/database/oracle/oracle-database/18/netag/configuring-oracle-connection-manager.html#GUID-3917FC5D-4B23-4752-85BA-39A88C4D13F8>

MySQL Usage

Oracle Traffic Director mode for Connection Manager can be potentially replaced by Amazon RDS Proxy for migration to Aurora MySQL

RDS Proxy simplifies connection management for Amazon RDS DB instances and clusters. It handles the network traffic between the client application and the database in an active way first by understanding the database protocol. Then RDS Proxy adjusts its behavior based on the SQL operations from user application and the result sets from the database.


RDS Proxy also reduces the memory and CPU overhead for the database connection management. The database needs less memory and CPU resources when applications open many simultaneous connections. RDS Proxy also doesn't require applications to close and reopen connections that stay idle for a long time. Similarly, it requires less application logic to reestablish connections in case of a database problem.

The infrastructure for RDS Proxy is highly available and deployed over multiple Availability Zones (AZs). The computation, memory, and storage for RDS Proxy are independent of RDS DB instances and Aurora DB clusters. This separation helps lower overhead on database servers, so that they can devote their resources to serving database workloads. The RDS Proxy compute resources are serverless, automatically scaling based on your database workload.

This topic has been covered in the following topic in this book: [Amazon RDS Proxy](#)

For more information see [Managing Connections with Amazon RDS Proxy](#)

Oracle Data Pump vs. MySQL mysqldump and mysql

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Non-compatible tool.

Oracle Usage

Oracle Data Pump is a utility for exporting/importing data to and from an Oracle database. It can be used to copy an entire database, entire schemas, or specific objects in a schema. Oracle Data Pump is commonly used as a part of a backup strategy for restoring individual database objects (specific records, tables, views, stored procedures, etc.) as opposed to snapshots or Oracle RMAN, which provides backup and recovery capabilities at the database level. By default (without using the sqlfile parameter during export), the “dump file” generated by Oracle Data Pump is binary (it cannot be read using a text editor).

Oracle Data Pump Supports:

- **Export Data from an Oracle database:** The Data Pump EXPDP command creates a binary dump file containing the exported database objects. Objects can be exported with data or with metadata only. Exports can be performed for specific timestamps or Oracle SCNs to ensure cross-object consistency.
- **Import Data to an Oracle database:** The Data Pump IMPDP command imports objects and data from a specific dump file created with the EXPDP command. The IMPDP command can filter on import (for example, only import certain objects) and remap object and schema names during import.

Notes:

- The term “Logical backup” refers to a dump file created by Oracle Data Pump.
- Both EXPDP and IMPDP can only read/write “dumpfiles” from file system paths that were pre-configured in the Oracle database as “directories”. During export/import, users must specify the logical “directory” name where the dump file should be created, not the actual file system path.

Examples

Use EXPDP to export the HR schema.

```
$ expdp system/**** directory=expdp_dir schemas=hr dumpfile=hr.dmp logfile=hr.log
```

Note: The command contains the credentials to run Data Pump, the logical Oracle directory name for the dump file location (which maps in the database to a physical file system location), the schema name to export, the dump file name, and log file name.

Use IMPDP to import the HR a schema and rename to HR_COPY.

```
$ impdp system/**** directory=expdp_dir schemas=hr dumpfile=hr.dmp logfile=hr.log  
REMAP_SCHEMA=hr:hr_copy
```

Note: The command contains the database credentials to run Data Pump, the logical Oracle directory for where the export dumpfile is located, the dump file name, the schema to export, the name for the dump file, the log file name, and the REMAP_SCHEMA parameter.

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sutil/oracle-data-pump.html#GUID-501A9908-BCC5-434C-8853-9A6096766B5A>

MySQL Usage

MySQL provides the native utilities mysqldump and mysqlimport for performing logical database exports and imports. The functionality is comparable to Oracle's Data Pump utility; however, in some use cases, the mysql connection utility is more equivalent to Oracle's Data Pump import tool (impdp). These utilities can be used to move data between two different databases or to create logical database backups.

In order to explain the difference between mysql and mysqlimport utilities, the equivalent Oracle reference will be used.

Use mysql utility to interact with the database like SQL*Plus. For import purposes, you can run it with all of the CREATE and INSERT commands to rebuild your schema and insert the data just like in Oracle.

The mysqlimport utility reads a data file (CSV file) and is equivalent to SQL*Loader. This utility is a reference to the LOAD DATA command. It is mostly used to move schema or some of the objects between clusters. You should use this utility if you have a data file (not a script) and you want to load it fast.

- mysqldump is equivalent to Oracle expdp
- mysql is equivalent to Oracle impdp

Amazon Aurora MySQL supports data export and import using mysqldump, mysqlimport, or mysql creation scripts. The binaries for all utilities must be installed on your local workstation or on an Amazon EC2 server.

After export, MySQL dump files created using mysqldump can be copied to an Amazon S3 bucket. Later, when the dump files are needed for database restore, they can be copied back to a desktop/server with a MySQL client (such as your workstation or an Amazon EC2 server) to use mysqlimport.

Notes:

- mysqldump creates consistent backups only if using the --single-transaction option.
- mysqldump does not block other users accessing the database (readers or writers).
- Unlike Data Pump, mysqldump files are plain-text.

Examples

Export data using mysqldump.

```
mysqldump --column-statistics=0 DATABASE_TO_RESTORE -h INSTANCE_ENDPOINT -P 3306 -u
USER_NAME -p > /local_path/backup-file.sql
```

RDS MySQL Only (not Aurora MySQL): If you are using MySQL 8.0 binaries when running the mysqldump, you must use the "column_statistics" flag set to 0.

Run an export and copy the backup file to an Amazon S3 bucket using a pipe and the AWS CLI.

```
mysqldump --column-statistics=0 DATABASE_NAME -h MYSQL_INSTANCE_ENDPOINT -P 3306 -u
USER_NAME -p > /local_path/backup-file.sql | aws s3 cp - s3://mysql-backups/mysql_bck-
$(date "+%Y-%m-%d-%H-%M-%S")
```

Import data using mysql.

```
mysql DB_NAME -h MYSQL_INSTANCE_ENDPOINT -P 3306 -u USER_NAME -p < /local_path/backup-
file.sql
```

Copy the output file from the local server to an Amazon S3 Bucket using the AWS CLI.

```
aws s3 cp /local_path/backup-file.sql s3://my-bucket/backup-$(date
"+%Y-%m-%d-%H-%M-%S")
```

Note: The `{-$(date "+%Y-%m-%d-%H-%M-%S")}` format is valid on Linux servers only.

Download the output file from the S3 bucket.

```
$ aws s3 cp s3://my-bucket/backup-2017-09-10-01-10-10 /local_path/backup-file.sql
```

Summary


Description	Oracle Data Pump	MySQL Dump
Export data to a local file	expdp system/**** schemas=hr dump-file=hr.dmp logfile=hr.log	mysqldump --column-statistics=0 DATABASE_TO_RESTORE -h INSTANCE_ENDPOINT -P 3306 -u USER_NAME -p > /local_path/backup-file.sql
Export data to a remote file	Create Oracle directory on remote storage mount or NFS directory called EXP_DIR Use export command:	mysqldump --column-statistics=0 DATABASE_NAME -h MYSQL_INSTANCE_ENDPOINT -P 3306 -u USER_NAME -p > /local_path/backup-file.sql aws s3 cp -

Description	Oracle Data Pump	MySQL Dump
	expdp system/**** schemas=hr directory=EXP_DIR dumpfile=hr.dmp logfile=hr.log	s3://mysql-backups/mysql_bck-\$(date +%Y-%m-%d-%H-%M-%S)
Import data to a new database with a new name	impdp system/**** schemas=hr dumpfile=hr.dmp logfile=hr.log REMAP_SCHEMA=hr:hr_copy TRANSFORM=OID:N	mysql DB_NAME -h MYSQL_INSTANCE_ENDPOINT -P 3306 -u USER_NAME -p </local_path/backup-file.sql

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>
- <https://dev.mysql.com/doc/refman/5.7/en/mysqlimport.html>
- <https://dev.mysql.com/doc/refman/5.7/en/mysql.html>

Oracle Flashback Database vs. MySQL Snapshots

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Storage level backup managed by Amazon RDS

Oracle Usage

Oracle Flashback Database is a special mechanism built into Oracle databases that helps protect against human errors by providing capabilities to revert the entire database back to a previous point in time using SQL commands. Flashback database implements a self-logging mechanism that captures all changes applied to a database and to data. Essentially, it stores previous versions of database modifications in the configured database "Fast Recovery Area".

When using Oracle flashback database, you can choose to restore an entire database to either a user-created restore point, a timestamp value, or to a specific System Change Number (SCN).

Examples

Create a database restore point to which you can flashback a database.

```
CREATE RESTORE POINT before_update GUARANTEE FLASHBACK DATABASE;
```

Flashback a database to a previously created restore point.

```
shutdown immediate;
startup mount;
flashback database to restore point before_update;
```

Flashback a database to a specific time.

```
shutdown immediate;
startup mount;
FLASHBACK DATABASE TO TIME "TO_DATE('01/01/2017','MM/DD/YY')";
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/rcmf/FLASHBACK-DATABASE.html#GUID-584AC79A-40C5-45CA-8C63-DED3BE3A4511>

MySQL Usage

Snapshots are the primary backup mechanism for Amazon Aurora databases. They are extremely fast and non-intrusive. You can take snapshots using the Amazon RDS Management Console or the AWS CLI. Unlike RMAN, there is no need for incremental backups. You can choose to restore your database to the exact time when a snapshot was taken or to any other point in time.

Amazon Aurora provides the following types of backups:

- **Automated Backups:** Can be enabled during Amazon Aurora instance created and to retain up to 35 days. They do not impact database performance.
- **Manual Backups:** You can create a snapshot at any time. There is no performance impact when taking snapshots of an Aurora database. Restoring data from snapshots requires creation of a new instance. Up to 100 manual snapshots are supported per database.

When using Aurora MySQL 5.6 compatible, a feature called "[Aurora Backtrack](#)" can be enabled. This feature is equivalent to Flashback Database option in Oracle.

This option applies to newly created MySQL-compatible Aurora database clusters and to MySQL-compatible clusters that have been restored from a backup. You must opt-in when you create or restore a cluster; you cannot enable it for a running cluster.

To backtrack your database using the CLI, the example below will backtrack to instance 86,400 seconds back (one day):

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster --backtrack-window 86400
```

To monitor the backtrack operation use:

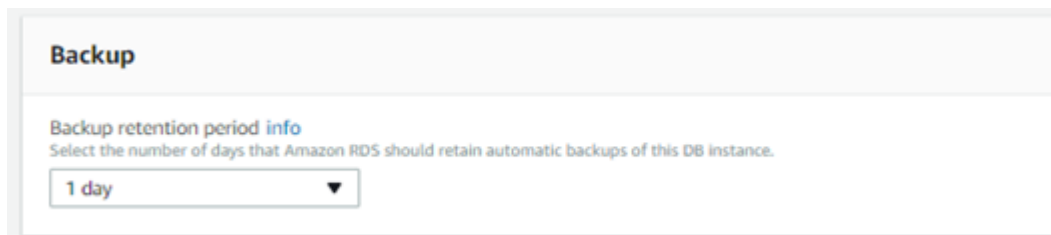
```
aws rds describe-db-cluster-backtracks --db-cluster-identifier sample-cluster
```

Examples

Enable Automatic Backups

Follow the steps below to enable Aurora automatic backups and configure the backup retention window as part of the database creation process. This process is equivalent to setting the Oracle RMAN backup retention policy using the “configure retention policy to recovery window of X days” command.

1. Navigate to the [RDS Instance Page](#).
2. If you are launching a new instance or modifying an existing instance, select the backup retention in days



Backup

Backup retention period info
Select the number of days that Amazon RDS should retain automatic backups of this DB instance.

1 day ▼

The following table identifies the default automatic backup time for each region:

Regio	Default Backup Window
US West (Oregon)	06:00-14:00 UTC
US West (N. California)	06:00-14:00 UTC
US East (Ohio)	03:00-11:00 UTC
US East (N. Virginia)	03:00-11:00 UTC
Asia Pacific (Mumbai)	16:30-00:30 UTC
Asia Pacific (Seoul)	13:00-21:00 UTC
Asia Pacific (Singapore)	14:00-22:00 UTC
Asia Pacific (Sydney)	12:00-20:00 UTC
Asia Pacific (Tokyo)	13:00-21:00 UTC
Canada (Central)	06:29-14:29 UTC
EU (Frankfurt)	20:00-04:00 UTC
EU (Ireland)	22:00-06:00 UTC
EU (London)	06:00-14:00 UTC
South America (São Paulo)	23:00-07:00 UTC
AWS GovCloud (US)	03:00-11:00 UTC

Perform a Manual Backup

Follow the steps below to perform a manual snapshot backup of an Aurora database. This process is equivalent to creating a full Oracle RMAN backup (“BACKUP DATABASE PLUS ARCHIVELOG”).

1. Navigate to the [RDS Databases Page](#).
2. Select an Aurora MySQL instance, click **Instance actions** and select **Take Snapshot**.

The screenshot shows the AWS RDS 'Databases' console. At the top, there are buttons for 'Group resources', 'Modify', 'Actions', 'Restore from S3', and 'Create database'. Below these is a search bar for 'Filter databases'. A table lists database instances. The instance 'mysql-aurora-playbook-instance-1' is selected. A context menu is open over this instance, showing options: 'Reboot', 'Delete', 'Failover', 'Take snapshot', and 'Start database activity stream'. The 'Take snapshot' option is highlighted.

DB identifier	Engine	Region & A
mysql-aurora-playbook	Aurora MySQL	eu-central-
mysql-aurora-playbook-instance-1	Writer Aurora MySQL	eu-central-

Restore a Database From a Snapshot

Follow the steps below to restore an Aurora database from a snapshot. This process is similar to the Oracle RMAN commands `RESTORE DATABASE` and `RECOVER DATABASE`. However, instead of running in place, restoring an Aurora database creates a new cluster.

1. Navigate to the [RDS System Snapshots](#) (link will refer to System snapshots but another tab can be used to view Manual snapshots).
2. Select the snapshot to restore, click **Actions** on the context menu, and select **Restore snapshot**. This action creates a new instance.

The screenshot shows the AWS RDS 'Snapshots' console. At the top, there are tabs for 'Manual', 'System', 'Shared with me', 'Public', 'Backup service', and 'Exports in Amazon S3'. The 'System' tab is selected. Below this is a search bar for 'Filter system snapshots'. A table lists system snapshots. The snapshot 'rds:mysql-aurora-playbook-2021-02-26-02-32' is selected. A context menu is open over this snapshot, showing options: 'Restore snapshot', 'Copy snapshot', 'Share snapshot', 'Migrate snapshot', 'Export to Amazon S3', and 'Delete snapshot'. The 'Restore snapshot' option is highlighted.

Snapshot name	DB instance or cluster	Created
<input checked="" type="checkbox"/> rds:mysql-aurora-playbook-2021-02-26-02-32	mysql-aurora-playbook	
<input type="checkbox"/> rds:pg-playbooks-2021-02-25-21-26	pg-playbooks	
<input type="checkbox"/> rds:oraplaybook-2021-02-25-21-03	oraplaybook	February 25, 2021, 9:0
<input type="checkbox"/> rds:mysql-aurora-playbook-2021-02-25-02-32	mysql-aurora-playbook	February 25, 2021, 2:3
<input type="checkbox"/> rds:pg-playbooks-2021-02-24-21-26	pg-playbooks	February 24, 2021, 9:2

3. The web page displays a wizard for creating a new Aurora database instance from the selected snapshot. Enter the required configuration options and click **Restore DB Instance**.

Restore a Database to a Point in Time

Follow the steps below to restore an Aurora MySQL database backup to a specific point in time. This process is similar to running the Oracle RMAN command `SET UNTIL TIME "TO_DATE('XXX')"` before running `RESTORE DATABASE` and `RECOVER DATABASE`.

1. Navigate to the [RDS Instance Page](#).
2. Select an Aurora instance, click **Instance actions**, and select **Restore to point in time**.

The screenshot shows the AWS Management Console 'Databases' page. At the top, there are buttons for 'Group resources', 'Modify', 'Actions', 'Restore from S3', and 'Create database'. A search bar labeled 'Filter databases' is present. Below, a table lists database instances. The instance 'mysql-aurora-playbook' is selected. The 'Actions' dropdown menu is open, showing options like 'Upgrade at next window', 'Add AWS Region', 'Add reader', 'Create cross-Region read replica', 'Create clone', 'Promote', 'Restore to point in time' (highlighted), 'Backtrack', and 'Add replica auto scaling'. The table columns include 'DB identifier', 'Name', and 'Region & Availability Zone'.

3. This process launches a new instance. Select the date and time to which you want to restore your database. The selected date and time must be within the configured backup retention for this instance.

AWS CLI Backup and Restore Operations

In addition to using the AWS web console to backup and restore an Aurora instance snapshot, you can also use the AWS CLI to perform the same actions. The CLI is especially useful for migrating existing automated Oracle RMAN scripts to an AWS environment. The following list highlights some CLI operations:

- Use `describe-db-cluster-snapshots` to view all current Aurora MySQL snapshots.
- Use `create-db-cluster-snapshot` to create a snapshot ("Restore Point").
- Use `restore-db-cluster-from-snapshot` to restore a new cluster from an existing database snapshot.
- Use `create-db-instance` to add new instances to the restored cluster.

```
aws rds describe-db-cluster-snapshots

aws rds create-db-cluster-snapshot --db-cluster-snapshot-identifier Snapshot_name
--db-cluster-identifier Cluster_Name

aws rds restore-db-cluster-from-snapshot --db-cluster-identifier NewCluster --
snapshot-identifier SnapshotToRestore --engine aurora-mysql

aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine
aurora-mysql --db-cluster-identifier NewCluster --db-instance-identifier new-
instance-nodeA --db-instance-class db.r4.large
```

- Use **restore-db-instance-to-point-in-time** to perform point-in-time recovery.

```
aws rds restore-db-cluster-to-point-in-time --db-cluster-identifier clustername-restore --source-db-cluster-identifier clustername --restore-to-time 2017-09-19T23:45:00.000Z
```

```
aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine aurora-mysql --db-cluster-identifier clustername-restore --db-instance-identifier newinstance-nodeA --db-instance-class db.r4.large
```


Summary

Description	Oracle	Amazon Aurora
Create a “restore point”	CREATE RESTORE POINT before_update GUARANTEE FLASHBACK DATABASE;	aws rds create-db-cluster-snapshot --db-cluster-snapshot-identifier Snapshot_name --db-cluster-identifier Cluster_Name
Configure flashback “retention period”	ALTER SYSTEM SET db_flashback_retention_target=2880;	Configure the “Backup retention window” setting using the AWS management console or AWS CLI.
Flashback database to a previous “restore point”	shutdown immediate; startup mount; flashback database to restore point before_update;	1. Create new cluster from a snapshot: aws rds restore-db-cluster-from-snapshot --db-cluster-identifier NewCluster --snapshot-identifier SnapshotToRestore --engine aurora-mysql 2. Add new instance to the cluster: aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine aurora-mysql --db-cluster-identifier clustername-restore --db-instance-identifier newinstance-nodeA --db-instance-class db.r4.large
Flashback database to a previous point in time	shutdown immediate; startup mount; FLASHBACK DATABASE TO TIME "TO_DATE ('01/01/2017','MM/DD/YY')";	In this example the cluster will be restore to 86400 seconds ago (1 day) aws rds modify-db-cluster --db-cluster-identifier sample-cluster --backtrack-window 86400

For more details, see:

- [rds](#)
- [Restoring a DB Instance to a Specified Time](#)
- [Restoring from a DB Snapshot](#)
- <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>

Oracle Flashback Table vs. MySQL Snapshots (Flashback Tables)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Storage level backup managed by Amazon RDS

Oracle Usage

Oracle Flashback Table is a data protection feature used to undo changes to a table and rewind it to a previous state (not from backup). While Flashback table operations are running, the affected tables are locked, but the rest of the database remains available.

Notes:

- If the structure of a table has been changed since the point of restore, the FLASHBACK will fail.
- Row movement must be enabled.
- The data to restore must be found in the undo (dba must manage the size and retention).
- A table can be restored to an System Change Number (SCN), Restore Point, or Timestamp.

Examples

Flashback a table using SCN (query V\$DATABASE to obtain the SCN).

```
SELECT CURRENT_SCN FROM V$DATABASE;
FLASHBACK TABLE employees TO SCN 3254648;
```

Flashback a table using a Restore Point (query V\$RESTORE_POINT to obtain restore points).

```
SELECT NAME, SCN, TIME FROM V$RESTORE_POINT;
FLASHBACK TABLE employees TO RESTORE POINT employees_year_update;
```

Flashback a table using a Timestamp (query V\$PARAMETER to obtain the undo_retention value).

```
SELECT NAME, VALUE/60 MINUTES_RETAINED
FROM V$PARAMETER
WHERE NAME = 'undo_retention';
FLASHBACK TABLE employees TO
TIMESTAMP TO_TIMESTAMP('2017-09-21 09:30:00', 'YYYY-MM-DD HH:MI:SS');
```

For more details, see [Performing Flashback and Database Point-in-Time Recovery](#)

MySQL Usage

Snapshots are the primary backup mechanism for Amazon Aurora databases. They are extremely fast and non-intrusive. You can take snapshots using the Amazon RDS Management Console or the AWS CLI. Unlike RMAN, there is no need for incremental backups. You can choose to restore your database to the exact time when a snapshot was taken or to any other point in time.

Amazon Aurora provides the following types of backups:

- **Automated Backups:** Can be enabled during Amazon Aurora instance created and to retain up to 35 days. They do not impact database performance.
- **Manual Backups:** You can create a snapshot at any time. There is no performance impact when taking snapshots of an Aurora database. Restoring data from snapshots requires creation of a new instance. Up to 100 manual snapshots are supported per database.

Examples

For examples, see [Flashback Database](#).

Summary


Description	Oracle	Amazon Aurora
Create a “restore point”	CREATE RESTORE POINT before_update GUARANTEE FLASHBACK DATABASE;	aws rds create-db-cluster-snapshot --db-cluster-snapshot-identifier Snapshot_name --db-cluster-identifier Cluster_Name
Configure flashback “retention period”	ALTER SYSTEM SET db_flashback_retention_target=2880;	Configure the “Backup retention window” setting using the AWS management console or using the AWS CLI.
Flashback table to a previous “restore point”	shutdown immediate; startup mount; flashback database to restore point before_update;	<ol style="list-style-type: none"> 1. Create new cluster from a snapshot: aws rds restore-db-cluster-from-snapshot --db-cluster-identifier NewCluster --snapshot-identifier SnapshotToRestore --engine aurora-mysql 2. Add new instance to the cluster: aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine aurora-mysql --db-cluster-identifier clustername-restore --db-instance-identifier newinstance-nodeA --db-instance-class db.r4.large 3. Use mysqldbexport and mysql to copy the table from the restored instance to the original instance

Description	Oracle	Amazon Aurora
Flashback table to a previous point in time	<pre>shutdown immediate; startup mount; FLASHBACK DATABASE TO TIME "TO_DATE ('01/01/2017','MM/DD/YY)";</pre>	<p>1. Create a new cluster from a snapshot and provide a specific point in time:</p> <pre>aws rds restore-db-cluster-to-point-in-time --db-cluster-identifier clusternamerestore --source-db-cluster-identifier clusternamerestore --restore-to-time 2017-09-19T23:45:00.000Z</pre> <p>2. Add a new instance to the cluster:</p> <pre>aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine aurora-mysql --db-cluster-identifier clusternamerestore --db-instance-identifier newinstance-nodeA --db-instance-class db.r4.large</pre> <p>3. Use mysqldbexport and mysql to copy the table from the restored instance to the original instance</p>

For more details, see:

- [rds](#)
- [Restoring a DB Instance to a Specified Time](#)
- [Restoring from a DB Snapshot](#)
- <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>

Oracle Recovery Manager (RMAN) vs. MySQL Snapshots (RMAN)

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Storage level backup managed by Amazon RDS

Oracle Usage

Oracle Recovery Manager (RMAN) is Oracle's primary backup and recovery tool. It provides its own scripting syntax and can be used to take full or incremental backups of an Oracle database. The following list identifies the types of backups.

- **Full RMAN Backup:** Take a full backup of an entire database or individual Oracle data files. For example, a level 0 full backup.
- **Differential Incremental RMAN Backup:** Performs a backup of all database blocks that have changed from the previous level 0 or 1 backup.
- **Cumulative Incremental RMAN Backup:** Perform a backup all of blocks that have changed from the previous level 0 backup.

Notes:

- RMAN supports online backups of an Oracle database if it has been configured to run in Archived Log Mode.
- RMAN backs up the following files:
 - Database data files.
 - Database control file.
 - Database parameter file.
 - Database Archived Redo Logs.

Examples

Use the RMAN CLI to connect to an Oracle database.

```
export ORACLE_SID=ORCL
rman target=/
```

Perform a full backup of the database and the database archived redo logs.

```
BACKUP DATABASE PLUS ARCHIVELOG;
```

Perform an incremental level 0 or level 1 backup of the database.

```
BACKUP INCREMENTAL LEVEL 0 DATABASE;
BACKUP INCREMENTAL LEVEL 1 DATABASE;
```

Restore a database.

```
RUN {
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
RESTORE DATABASE;
RECOVER DATABASE;
ALTER DATABASE OPEN;
}
```

Restore a specific pluggable database (Oracle 12c).

```
RUN {
ALTER PLUGGABLE DATABASE pdbA, pdbB CLOSE;
RESTORE PLUGGABLE DATABASE pdbA, pdbB;
RECOVER PLUGGABLE DATABASE pdbA, pdbB;
ALTER PLUGGABLE DATABASE pdbA, pdbB OPEN;
}
```

Restore a database to a specific point in time.

```
RUN {
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
SET UNTIL TIME "TO_DATE('20-SEP-2017 21:30:00','DD-MON-YYYY HH24:MI:SS')";
RESTORE DATABASE;
RECOVER DATABASE;
ALTER DATABASE OPEN RESETLOGS;
}
```

List all current database backups created with RMAN.

```
LIST BACKUP OF DATABASE;
```

For more details, see [Database Backup and Recovery User's Guide](#)

MySQL Usage

Snapshots are the primary backup mechanism for Amazon Aurora databases. They are extremely fast and non-intrusive. You can take snapshots using the Amazon RDS Management Console or the AWS CLI. Unlike RMAN, there is no need for incremental backups. You can choose to restore your database to the exact time when a snapshot was taken or to any other point in time.

Amazon Aurora provides the following types of backups:

- **Automated Backups:** Can be enabled during Amazon Aurora instance created and to retain up to 35 days. They do not impact database performance.
- **Manual Backups:** You can create a snapshot at any time. There is no performance impact when taking snapshots of an Aurora database. Restoring data from snapshots requires creation of a new instance. Up to 100 manual snapshots are supported per database.

RDS MySQL Only (not Aurora MySQL): As of MySQL 8.0.21 redo logging can be enabled and disabled using ALTER INSTANCE {ENABLE|DISABLE} INNODB REDO_LOG syntax. This functionality is intended for loading data into a new MySQL instance. Disabling redo logging helps speed up data loading by avoiding redo log writes. The new INNODB_REDO_LOG_ENABLE privilege permits enabling and disabling redo logging. The new Innodb_redo_log_enabled status variable permits monitoring redo logging status. See [Disabling Redo Logging](#).

Examples

For examples, see [Flashback Database](#).

Summary


Description	Oracle	Amazon Aurora
Scheduled backups	Create DBMS_SCHEDULER job that will execute your RMAN script on a scheduled basis.	Automatic
Manual full database backups	BACKUP DATABASE PLUS ARCHIVELOG;	Use Amazon the RDS dashboard or the AWS CLI to take a snapshot on the cluster: <code>aws rds create-db-cluster-snapshot --db-cluster-snapshot-identifier Snapshot_name --db-cluster-identifier Cluster_Name</code>
Restore database	<pre> RUN { SHUTDOWN IMMEDIATE; STARTUP MOUNT; RESTORE DATABASE; RECOVER DATABASE; ALTER DATABASE OPEN; } </pre>	<p>Create new cluster from a cluster snapshot:</p> <pre>aws rds restore-db-cluster-from-snapshot --db-cluster-identifier NewCluster --snapshot-identifier SnapshotToRestore --engine aurora-mysql</pre> <p>Add a new instance to the new/restored cluster:</p> <pre>aws rds create-db-instance --region us-east-1 --db-subnet-group default --engine aurora-myql --db-cluster-identifier NewCluster --db-instance-identifier newinstance-nodeA --db-instance-class db.r4.large</pre>
Incremental differential	<pre> BACKUP INCREMENTAL LEVEL 0 DATABASE; BACKUP INCREMENTAL LEVEL 1 DATABASE; </pre>	N/A
Incremental cumulative	<pre> BACKUP INCREMENTAL LEVEL 0 CUMULATIVE DATABASE; BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE; </pre>	N/A
Restore database to a specific point-in-time	<pre> RUN { SHUTDOWN IMMEDIATE; STARTUP MOUNT; SET UNTIL TIME "TO_DATE('19-SEP-2017 23:45:00','DD-MON-YYYY </pre>	<p>Create new cluster from a cluster snapshot by custom time to restore:</p> <pre>aws rds restore-db-cluster-to-point-in-time --db-</pre>

Description	Oracle	Amazon Aurora
	<pre>HH24:MI:SS)"; RESTORE DATABASE; RECOVER DATABASE; ALTER DATABASE OPEN RESETLOGS; }</pre>	<pre>cluster-identifier clustername-restore --source-db- cluster-identifier clustername --restore-to-time 2017-09-19T23:45:00.000Z</pre> <p>Add a new instance to the new/restored cluster:</p> <pre>aws rds create-db-instance --region us-east-1 --db- subnet-group default --engine aurora-mysql --db- cluster-identifier clustername-restore --db-instance- identifier newinstance-nodeA --db-</pre>
Backup database Archive logs	<pre>BACKUP ARCHIVELOG ALL;</pre>	N/A
Delete old data-base Archive logs	<pre>CROSSCHECK BACKUP; DELETE EXPIRED BACKUP;</pre>	N/A
Restore a single Pluggable data-base (12c)	<pre>RUN { ALTER PLUGGABLE DATABASE pdb1, pdb2 CLOSE; RESTORE PLUGGABLE DATABASE pdb1, pdb2; RECOVER PLUGGABLE DATABASE pdb1, pdb2; ALTER PLUGGABLE DATABASE pdb1, pdb2 OPEN; }</pre>	<p>Create new cluster from a cluster snapshot:</p> <pre>aws rds restore-db-cluster-from-snapshot --db- cluster-identifier NewCluster --snapshot-identifier SnapshotToRestore --engine aurora-mysql</pre> <p>Add a new instance to the new/restored cluster:</p> <pre>aws rds create-db-instance --region us-east-1 --db- subnet-group default --engine aurora-mysql --db- cluster-identifier clustername-restore --db-instance- identifier newinstance-nodeA --db-instance-class db.r4.large</pre> <p>Use mysqldump and mysql to copy the database to the original instance:</p> <pre>mysqldump --column-statistics=0 DATABASE_ TO_RESTORE -h RESTORED_INSTANCE_ ENDPOINT -P 3306 -u USER_NAME -p > /local_ path/backup-file.sql</pre> <pre>mysql DB_NAME -h MYSQL_INSTANCE_ ENDPOINT -P 3306 -u USER_NAME -p < /local_ path/backup-file.sql</pre> <div data-bbox="906 1682 1434 1899" style="background-color: #e0e0e0; padding: 10px; border: 1px solid #ccc;"> <p>RDS MySQL Only (not Aurora MySQL): if you are using MySQL 8.0 binaries when running the mysqldump , you must use the "column_statistics" flag set to 0</p> </div>

For more details, see:

- [rds](#)
- [Restoring a DB Instance to a Specified Time](#)
- [Restoring from a DB Snapshot](#)
- [RestoreFromSnapshot.](#)
- <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>

Oracle SQL*Loader vs. MySQL mysqlimport and LOAD DATA

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Non compatible tool

Oracle Usage

SQL*Loader is powerful utility that imports data from external files into database tables. It has strong parsing engine with few limitations on data formats.

SQL*Loader can be used with or without a control file. A control file enables handling more complicated load environments. For simpler loads, use SQL*Loader without a control file (also referred to as SQL*Loader Express).

The outputs of SQL*Loader include the imported database data, a log file, a bad file (rejected records), and a discard file (if enabled).

Examples

Oracle SQL*Loader is well suited for large databases with a limited number of objects. The process of exporting from a source database and loading to a target database is very specific to the schema. The following example creates sample schema objects, exports from a source, and loads into a target database.

Create a sample source table.

```
CREATE TABLE customer_0 TABLESPACE users
AS SELECT rownum id, o.* FROM all_objects o, all_objects x where rownum <= 1000000;
```

On the target Amazon RDS instance, create a destination table for the loaded data.

```
CREATE TABLE customer_1 TABLESPACE users
AS select 0 as id, owner, object_name, created from all_objects where 1=2;
```

The data is exported from the source database to a flat file with delimiters. This example uses SQL*Plus. For your data, you will likely need to generate a script that does the export for all the objects in the database.

```
alter session set nls_date_format = 'YYYY/MM/DD HH24:MI:SS';
set linesize 800
HEADING OFF FEEDBACK OFF array 5000 pagesize 0
spool customer_0.out
SET MARKUP HTML PREFORMAT ON SET COLSEP ',' SELECT id, owner, object_name,created FROM
customer_0;
spool off
```

Create a control file describing the data. Depending on the data, you may need to build a script that provides this functionality.

```
cat << EOF > sqlldr_1.ctl
LOAD DATA
```

```

INFILE customer_0.out
into table customer_1
APPEND
fields terminated by "," optionally enclosed by '"'
(id                POSITION(01:10)          INTEGER EXTERNAL,
owner              POSITION(12:41)          CHAR,
object_name        POSITION(43:72)          CHAR,
created            POSITION(74:92)          date "YYYY/MM/DD HH24:MI:SS")

```

Import the data using SQL*Loader. Use the appropriate username and password for the target database.

```

sqlldr cust_dba@targetdb control=sqlldr_1.ctl BINDSIZE=10485760 READSIZE=10485760 ROWS-
S=1000

```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/sutil/oracle-sql-loader.html#GUID-8D037494-07FA-4226-B507-E1B2ED10C144>

MySQL Usage

There are two relevant options when looking for a replacement for the Oracle SQL*Loader utility:

- **MySQL Import:** using an export file similar to a control file.
- **Load from S3 File:** using a table-formatted file on S3 and loading it into a MySQL database.

mysqlimport is a good option when it's required to use a tool from another server or a client. The LOAD DATA command can be combined with metadata tables and EVENT objects to schedule loads.

For more details, see

- [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket](#)
- <https://dev.mysql.com/doc/refman/5.7/en/mysqlimport.html>

Configuration



Oracle Upgrades vs. Aurora for MySQL Upgrades

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
N/A	N/A	N/A	

Oracle Usage

As a Database Administrator, from time to time a database upgrade is required, it can be either for security fix, but, or a new database feature.

The Oracle upgrades are divided into two different types of upgrades, minor and major.

This topic will outline the differences between the procedure to execute upgrades on your Oracle databases today and how you will run those upgrades post migrating to RDS running Aurora.

The regular presentation of Oracle versions is combined of 4 numbers divided by dots (sometimes you will see a a fifth number)

Either way, major or minor upgrades, the first step to initiate the processes mentioned above would be to install the new Oracle software on the database server, and of course before upgrading a production database to have an extensive amount of testing with the applications using the database to upgrade.

Oracle 18c introduces Zero-Downtime Database Upgrade to automate Database upgrade and potentially eliminate application downtime during the this process.

In order to understand the versions, lets use the following example 11.2.0.4.0

What those digits above means:

- 11 - this is the major database version
- 2 - this is the database maintenance version
- 0 - Application server version
- 4 - Component specific version
- 0 - Platform specific version

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/upgrd/about-oracle-database-release-numbers.html#GUID-1E2F3945-C0EE-4EB2-A933-8D1862D8ECE2>

Oracle allows the users to set the compatibility level of the database to control the features and some behaviors.

This is being done using the COMPATIBLE parameter, the value for this parameter can be fetched using the following query

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'compatible';
```

Upgrade process

In general, the process to execute major or minor upgrades is the same, minor version upgrade has less steps but overall the process is very similar.

Major upgrade referring to upgrades of the version number in the Oracle version, in the example above "11", the minor upgrade refers to any of the following numbers in the Oracle version, in the example above these will be "2.0.4.0".

Major upgrades are mostly being done in order to gain many new useful features being released between those versions, while minor upgrades are focused on bug and security fixes.

Upgrades can be done using the Oracle upgrade tools or manually.

Oracle tools will perform the following steps and might ask for some inputs or fixes from the user along the process.

The upgrade steps are:

- Upgrade operation type - user will choose either Oracle database upgrade or move database between Oracle software installations
- Database selection - user will select the database to upgrade and the Oracle software to use for this database
- Prerequisite checks - Oracle tools will let the user choose what to do with all issues found and their severity
- Upgrade options - Oracle will let the user to pick his practices to do the upgrade, options like: recompilation and parallelism for those, timezone upgrade, statistics gathering, and more.
- Management options - user will choose to connect / configure Oracle management solutions to the database
- Move database files - user will choose if a data file movement is required to a new device/path
- Network configuration - Oracle listener configurations
- Recovery options - user to define Oracle backup solutions or using his own
- Summary - a report of all options were selected in previous steps to present before the upgrade
- Progress - monitor and present the upgrade status
- Results - a post upgrade summary

For the manual process, we won't cover all actions in this topic, as there are many steps and commands to execute.

In overall, the eleven steps mentioned above will be divided into many sub-steps and tasks to executing

For the entire manual option, please see: <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/upgrd/example-manual-upgrade-windows-non-cdb-11203-to-122.html#GUID-30F3DC9C-141A-47DC-9B83-6D0C395E565C>

MySQL Usage

After migrating your databases to RDS running Aurora for MySQL, you will still need to upgrade your database instance from time to time, for the same reasons you have done it in the past like new features, bugs and security fixes.

In a managed service like RDS, the upgrade process is much easier and simpler compare to the on-prem Oracle process.

To determine the current Aurora for MySQL version being used, you can use the following aws cli command:

```
aws rds describe-db-engine-versions --engine aurora-mysql --query '*[].[EngineVersion]' --output text --region your-AWS-Region
```

This can also be queried from the database, using the following queries:

```
SELECT AURORA_VERSION();
```

In an Aurora MySQL version number scheme, for example 2.08.1, first digit represents the major version. Aurora MySQL version 1 is compatible with MySQL 5.6 and Aurora MySQL version 2 is compatible with MySQL 5.7. All Aurora and MySQL versions mapping can be found in here: <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Updates.20Updates.html>

AWS doesn't apply major version upgrades on RDS Aurora automatically. Major version upgrades contains new features and functionality which often involves system table and other code changes. These changes may not be backward-compatible with previous versions of the database so applications testing is highly recommended.

Applying automatic minor upgrades can be set by configuring the RDS instance to allow it.

You can use the following aws cli command (linux) to determine the current automatic upgrade minor versions.

```
aws rds describe-db-engine-versions --output=table --engine mysql --engine-version minor-version --region region
```

Note: If no results returned, there is no automatic minor version upgrade available and scheduled.

When enabled, the instance will be automatically upgraded during the scheduled maintenance window.

If you want to upgrade your cluster to a compatible cluster, you can do so by running an upgrade process on the cluster itself. This kind of upgrade is an in-place upgrade, in contrast to upgrades that you do by creating a new cluster. The upgrade is relatively fast because it doesn't require copying all your data to a new cluster volume. In-place upgrade preserves the endpoints and set of DB instances for your cluster.

To verify application compatibility, performance and maintenance procedures for the upgraded cluster, you can perform a simulation of the upgrade by doing following

- Cloning a cluster.
- Perform an in-place upgrade of the cloned cluster.
- Test applications, performance and so on, using the cloned cluster.
- Resolve any issues, adjust your upgrade plans to account for them.
- Once all the testing looks good, you can perform the in-place upgrade for your production cluster.

For major upgrades, this is the recommended

- Check for open XA transactions by executing the XA RECOVER statement. Commit or Rollback the XA transactions before starting the upgrade.
- Check for DDL statements by executing a SHOW PROCESSLIST statement and looking for CREATE, DROP, ALTER, RENAME, and TRUNCATE statements in the output. Allow all DDLs to finish before starting the upgrade.
- Check for any uncommitted rows by querying the INFORMATION_SCHEMA.INNODB_TRX table. The table contains one row for each transaction. Let the transaction complete or shut down applications that are submitting these changes.

Aurora MySQL performs a major version upgrade in multiple steps. As each step begins, Aurora MySQL records an event. You can monitor the current status and events as they occur on the Events page in the RDS console.

Aurora performs a series of checks before beginning the upgrade process. If any issues are detected during these checks, resolve the issue identified in the event details and restart the upgrade process.

Aurora takes the cluster offline, performs a similar set of tests as in the previous step. If no new issues are identified, then Aurora moves with the next step. If any issues are detected during these checks, resolve the issue identified in the event details and restart the upgrade process again.

Aurora backs up the MySQL cluster by creating a snapshot of the cluster volume.

Aurora clones the cluster volume. If any issues are encountered during the upgrade, Aurora reverts to the original data from the cloned cluster volume and brings the cluster back online.

Aurora performs a clean shutdown and it rolls back any uncommitted transactions.

Aurora upgrades the engine version. It installs the binary for the new engine version and uses the writer DB instance to upgrade your data to new to MySQL compatible format. During this stage, Aurora modifies the system tables and performs other conversions that affect the data in your cluster volume.

The upgrade process is completed. Aurora records a final event to indicate that the upgrade process completed successfully. Now DB cluster is running the new major version.

Upgrade can be done through the AWS Console or AWS cli.

Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose Databases, and then choose the DB cluster that you want to upgrade.
3. Choose Modify. The Modify DB cluster page appears.
4. For DB engine version, choose the new version.
5. Choose Continue and check the summary of modifications.
6. To apply the changes immediately, choose Apply immediately. Choosing this option can cause an outage in some cases. For more information, see [Modifying an Amazon Aurora DB cluster](#).
7. On the confirmation page, review your changes. If they are correct, choose Modify Cluster to save your changes.

Or choose Back to edit your changes or Cancel to cancel your changes.

AWS CLI

To upgrade the major version of an Aurora MySQL DB cluster, use the AWS CLI `modify-db-cluster` command with the following required parameters:

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine aurora-mysql \
--engine-version 5.7.mysql_aurora.2.09.0 \
--allow-major-version-upgrade \
--apply-immediately
```

For Windows:


```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine aurora-mysql ^
--engine-version 5.7.mysql_aurora.2.09.0 ^
--allow-major-version-upgrade ^
--apply-immediately
```

Summary

Phase	Oracle Step	Aurora for MySQL
Prerequisite	Install new Oracle software	N/A
Prerequisite	Upgrade operation type	N/A
Prerequisite	Database selection	Select right RDS instance
Prerequisite	Prerequisite checks	Commit or rollback uncommitted transactions
Prerequisite	Upgrade options	N/A
Prerequisite	Management options (optional)	N/A
Prerequisite	Move database files (optional)	N/A
Prerequisite	Network configuration (optional)	N/A
Prerequisite	Recovery options	N/A
Prerequisite	Summary	N/A
Prerequisite	Perform a database backup	Run RDS instance backup
Prerequisite	Stop application and connection	Same
Execution	Progress	Can be reviewed from the console
Post-upgrade	Results	Can be reviewed from the console
Post-upgrade	Test applications again the new upgraded database	Same
Production deployment	Re-run all steps in a production environment	Same

For additional details, see: <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Updates.Upgrading.html>

Oracle Alert.log and logs vs. MySQL Error Log via Amazon RDS Console

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Use Event Notifications Subscription with Amazon Simple Notification Service (SNS)

Oracle Usage

The primary Oracle error log file is the Alert Log. It contains verbose information about database activity including informational messages and errors. Each event includes a timestamp indicating when the event occurred. The Alert Log filename format is alert<sid>.log.

The Alert Log is the first place to look when troubleshooting or investigating errors, failures, and other messages indicating a potential database problem. Common events logged in the Alert Log include:

- Database startup or shutdown.
- Database redo log switch.
- Database errors and warnings, which begin with "ORA-" followed by an Oracle error number.
- Network and connection issues.
- Links for a detailed trace files about specific database events.

The Oracle Alert Log can be found inside the database Automatic Diagnostics Repository (ADR), which is a hierarchical file-based repository for diagnostic information: \$ADR_BASE/diag/rdbms/{DB-name}/{SID}/trace.

In addition, several other Oracle server components have unique log files such as the database listener and the Automatic Storage Manager (ASM).

Examples

The following screenshot displays partial contents of the Oracle database Alert Log File.

```
Sun Sep 03 13:27:23 2017
Starting ORACLE instance (normal)
***** Large Pages Information *****
Per process system memlock (soft) limit = 64 KB

Total Shared Global Region in Large Pages = 0 KB (0%)

Large Pages used by this instance: 0 (0 KB)
Large Pages unused system wide = 0 (0 KB)
Large Pages configured system wide = 0 (0 KB)
Large Page size = 2048 KB
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/monitoring-the-database.html#GUID-E5F89E8E-7FBC-47DD-BA5D-96AFD9CE4BC7>

MySQL Usage

MySQL provides detailed logging and reporting of errors that occur during the database and connected sessions life cycle. In an Amazon Aurora deployment, these informational and error messages are accessible using the Amazon RDS console.

MySQL vs. Oracle Error Codes

Oracle	MySQL
ORA-00001: unique constraint (string.string) violated	Error [1062][23000]: Duplicate entry 'value' for key 'column'

For more details, see [Server Error Message Reference](#)

Error Log Types

MySQL provides several type of logs:

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping mysqld.
General query log	Established client connections and statements received from clients.
Binary log	Statements that change data (also used for replication).
Relay log	Data changes received from a replication master server.
Slow query log	Queries that took more than long_query_time seconds to execute.
DDL log (meta-data log)	Meta-data operations performed by DDL statements.

For more information, see <https://dev.mysql.com/doc/refman/5.7/en/server-logs.html>

Examples

Access the MySQL error log using the Amazon RDS/Aurora Management Console:

1. Navigate to [Databases](#) in the RDS Service of the AWS Console.

Logs (402)			
<input type="text" value="Filter name"/> < 1 2 3 4 5 6 7 ... 81 >			
	Name	Last written	Size
<input type="radio"/>	error/mysql-error-running.log	Mon Feb 26 11:00:00 GMT+200 2018	10.6 kB
<input type="radio"/>	error/mysql-error-running.log.2018-02-09.19	Fri Feb 09 21:30:00 GMT+200 2018	93 kB
<input type="radio"/>	error/mysql-error-running.log.2018-02-09.20	Fri Feb 09 21:40:00 GMT+200 2018	4.4 kB
<input type="radio"/>	error/mysql-error-running.log.2018-02-09.21	Fri Feb 09 22:30:00 GMT+200 2018	18.1 kB
<input type="radio"/>	error/mysql-error-running.log.2018-02-09.22	Fri Feb 09 23:30:00 GMT+200 2018	22.5 kB

2. Click the instance name.
3. Search for 'Logs & events' tab and select the log to inspect. For example, select the log during the hour the data was experiencing problems. The following screen shot displays partial contents of a MySQL database error log as viewed from the Amazon RDS Management Console.

Viewing Log: error/mysql-error-running.log.2018-02-09.19 (93 kB)

text: background:

```

2018-02-09T19:01:48.847682Z 0 [Warning] 'NO_AUTO_CREATE_USER' sql mode was not set.
2018-02-09T19:01:48.848471Z 0 [Warning] Insecure configuration for --secure-file-priv:
Location is accessible to all OS users. Consider choosing a different directory.
2018-02-09T19:01:48.848497Z 0 [Note] /rdsdbbin/oscar/bin/mysqld (mysqld 5.7.12-log)
starting as process 5478 ...
2018-02-09T19:01:48.881215Z 0 [Warning] InnoDB: Setting innodb_checksums to OFF is
DEPRECATED. This option may be removed in future releases. You should set
innodb_checksum_algorithm=NONE instead.
2018-02-09T19:01:48.881317Z 0 [Note] InnoDB: PUNCH HOLE support not available
2018-02-09T19:01:48.881329Z 0 [Note] InnoDB: Mutexes and rw_locks use GCC atomic
builtins
2018-02-09T19:01:48.881333Z 0 [Note] InnoDB: Uses event mutexes
2018-02-09T19:01:48.881336Z 0 [Note] InnoDB: GCC builtin __atomic_thread_fence() is
used for memory barrier
2018-02-09T19:01:48.881340Z 0 [Note] InnoDB: Compressed tables use zlib 1.2.3
2018-02-09T19:01:48.882024Z 0 [Note] InnoDB: == Add SYNC_FAST DDL...
2018-02-09T19:01:48.884857Z 0 [Note] InnoDB: Number of pools: 1
2018-02-09T19:01:48.888837Z 0 [Note] InnoDB: Using CPU crc32 instructions
2018-02-09T19:01:48.890231Z 0 [Note] InnoDB: Initializing buffer pool, total size =

```


MySQL Error Log Configuration

Several parameters specify the location of the MySQL log and errors files. The following table identifies common Amazon Aurora configuration options.

Parameter	Description
<code>log_error</code>	Sets the file name and path for the Error log. Modifiable via an Aurora DatabaseParameter Group.
<code>log_error_verbosity</code>	Sets the message levels that are logged (Error, warning, note messages, etc....). Modifiable via an Aurora Database Parameter Group.
<code>USE SLOW LOG</code>	Sets the minimum execution time above which statements are logged (ms). Modifiable using an Aurora Database Parameter Group.

Note: Modifications of certain parameters, such as `log_error` are disabled for Aurora MySQL instances.

Oracle SGA & PGA Memory Sizing vs. MySQL Memory Buffers

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Different cache names, similar usage

Oracle Usage

An Oracle instance allocates several individual “pools” of server RAM used as various caches for the database. These include the Buffer Cache, Redo Buffer, Java Pool, Shared Pool, Large Pool, and others. The caches reside in the System Global Area (SGA) and are shared across all Oracle sessions.

In addition to the SGA, each Oracle session is granted an additional area of memory for session-private operations (sorting, private SQL cursors elements, etc.) called the Private Global Area (PGA).

Cache size can be controlled for individual caches or globally, and automatically, by an Oracle database. Setting a unified “memory size” parameter enables Oracle to automatically manage individual cache sizes.

- All Oracle memory parameters are set using the ALTER SYSTEM command.
- Some changes to memory parameters require an instance restart.

Some of the common Oracle parameters that control memory allocations include:

- **db_cache_size:** The size of the cache used for database data.
- **log_buffer:** The cache used to store Oracle redo log buffers until they are written to disk.
- **shared_pool_size:** The cache used to store shared cursors, stored procedures, control structures, and other structures.
- **large_pool_size:** The caches used for parallel queries and RMAN backup/restore operations.
- **Java_pool_size:** The caches used to store Java code and JVM context.

While these parameters can be configured individually, most database administrators choose to let Oracle automatically manage RAM. Database administrators configure the overall size of the SGA, and Oracle sizes individual caches based on workload characteristics.

- **sga_max_size:** Specifies the hard-limit maximum size of the SGA.
- **sga_target:** Sets the required soft-limit for the SGA and the individual caches within it.

Oracle also allows control over how much private memory is dedicated for each session. Database Administrators configure the total size of memory available for all connecting sessions, and Oracle allocates individual dedicated “chunks” from the total amount of available memory for each session.

- **pga_aggregate_target:** A soft-limit controlling the total amount of memory available for all sessions combined.
- **pga_aggregate_limit:** A hard-limit for the total amount of memory available for all sessions combined (Oracle 12c only) .

In addition, instead of manually configuring the SGA and PGA memory areas, you can also configure one overall memory limit for both the SGA and PGA and let Oracle automatically balance memory between the various memory pools. This behavior is enabled using the `memory_target` and `memory_max_target` parameters.

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/memory-architecture.html#GUID-913335DF-050A-479A-A653-68A064DCCA41>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/tgdba/database-memory-allocation.html#GUID-E9265077-B296-485A-BC2C-0AF55762D1EC>

MySQL Usage

Like other databases, MySQL uses different memory buffers for different purposes. In MySQL, there are several storage engines that use different memory buffers. This section refers to InnoDB only.

MySQL provides control over how server RAM is allocated. Some of the most important MySQL memory parameters include:

Memory Pool Parameter	Description
<code>innodb_buffer_pool_size</code>	The memory area where InnoDB caches table and index data.
<code>optimizer_trace_max_mem_size</code>	Buffer for optimizer traces.
<code>binlog_cache_size</code>	The size of the cache holding changes to the binary log during a transaction.
<code>host_cache_size</code>	Buffer area to store data on connections.
<code>innodb_ft_cache_size</code>	Very similar to <code>innodb_buffer_pool_size</code> but only for data related to FULL_TEXT indexes.
<code>stored_program_cache</code>	Cached stored routines per connection.
<code>sort_buffer_size</code>	Size of sort buffers used to sort data during creation of an InnoDB index.
Total memory available for MySQL Cluster	Controlled by selecting the “DB Instance Class” during instance creation:

Memory Pool Parameter	Description
	<p>Instance specifications</p> <p>Estimate your monthly costs for the DB Instance using the AWS Simple Monthly Calculator.</p> <p>DB engine Aurora - compatible with MySQL 5.7.12</p> <p>DB instance class info</p> <ul style="list-style-type: none"> db.t2.small — 1 vCPU, 2 GiB RAM db.t2.small — 1 vCPU, 2 GiB RAM db.t2.medium — 2 vCPU, 4 GiB RAM db.r3.large — 2 vCPU, 15.25 GiB RAM db.r3.xlarge — 4 vCPU, 30.5 GiB RAM db.r3.2xlarge — 8 vCPU, 61 GiB RAM db.r3.4xlarge — 16 vCPU, 122 GiB RAM db.r3.8xlarge — 32 vCPU, 244 GiB RAM db.r4.large — 2 vCPU, 15.25 GiB RAM db.r4.xlarge — 4 vCPU, 30.5 GiB RAM db.r4.2xlarge — 8 vCPU, 61 GiB RAM db.r4.4xlarge — 16 vCPU, 122 GiB RAM db.r4.8xlarge — 32 vCPU, 244 GiB RAM db.r4.16xlarge — 64 vCPU, 488 GiB RAM

Note: Cluster-level parameters such as `innodb_buffer_pool_size` and `binlog_cache_size` are configured using “parameter groups” in the Amazon RDS Management Console.

Examples

View the configured values for database parameters.

```
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
SHOW VARIABLES LIKE 'binlog_cache_size';
SHOW VARIABLES LIKE 'stored_program_cache';
```

View the configured values for all database parameters.

```
SELECT * FROM information_schema.GLOBAL_VARIABLES
```

Use the `SET SESSION` command to modify the value of parameters that support session-specific settings. Changing the value for one session has no effect on other sessions.

```
SET SESSION sort_buffer_size = 1000000;
```

Summary


Note: Use the table below as a general reference only. Functionality may not be identical across Oracle and MySQL.

Description	Oracle	MySQL
Memory for caching table data	db_cache_size	innodb_buffer_pool_size
Memory for transaction log records	log_buffer	binlog_cache_size
Memory for parallel queries	large_pool_size	N/A
Java code and JVM	Java_pool_size	N/A
Maximum amount of physical memory available for the Instance	sga_max_size or memory_max_size	Configured via the Amazon RDS/Aurora Instance class For example: db.r3.large: 15.25GB db.r3.xlarge: 30.5GB Etc.
Total amount of private memory for all sessions	pga_aggregate_target + pga_aggregate_limit	max_digest_length
View values for all database parameters	SELECT * FROM v\$parameter;	SELECT * FROM information_schema.GLOBAL_VARIABLES
Configure a session-level parameter	ALTER SESSION SET ...	SET SESSION ...
Configure instance-level parameter	ALTER SYSTEM SET ...	Configured via "Parameter Groups" in the Amazon RDS Management Console.

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html>
- <https://dev.mysql.com/doc/refman/5.7/en/memory-use.html>

Oracle Instance Parameters vs. MySQL Aurora Parameter Groups

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Use Cluster and Database/Cluster Parameters

Oracle Usage

Oracle Instance and database-level parameters can be configured using the ALTER SYSTEM command. Certain parameters can be configured dynamically and take immediate effect while other parameters require an instance restart.

- All Oracle instance and database-level parameters are stored in a binary file known as the Server Parameter file (SPFILE).
- The binary SPFILE can be exported to a text file using the following command: CREATE PFILE = 'my_init.ora' FROM SPFILE = 's_params.ora';

When modifying parameters, you can choose the persistence of the changed values with one of the three following options:

- Make the change applicable only after a restart by specifying scope=spfile.
- Make the change dynamically, but not persistent, after a restart by specifying scope=memory.
- Make the change both dynamically and persistent by specifying scope=both.

Examples

Use the ALTER SYSTEM SET command to configure a value for an Oracle parameter.

```
ALTER SYSTEM SET QUERY_REWRITE_ENABLED = TRUE SCOPE=BOTH;
```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/initialization-parameters-2.html#GUID-FD266F6F-D047-4EBB-8D96-B51B1DCA2D61>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/changing-parameter-values-in-a-parameter-file.html#GUID-4C578B21-DE2B-4210-8EB7-EF28D36CC1CB>

MySQL Usage

When running MySQL databases as Amazon Aurora Clusters, Parameter Groups are used to change to cluster-level and database-level parameters.

Most of the MySQL parameters are configurable in an Amazon Aurora MySQL cluster, but some are disabled and cannot be modified. Since Amazon Aurora clusters restrict access to the underlying operating system, modification to MySQL parameters must be made using Parameter Groups.

Amazon Aurora is a cluster of database instances and, as a direct result, some of the MySQL parameters apply to the entire cluster while other parameters apply only to a particular database instance.

Aurora MySQL Parameter Class	Controlled Via
<p>Cluster-level parameters Single cluster parameter group per Amazon Aurora Cluster</p>	<p>Managed using cluster parameter groups. For example: aurora_load_from_s3_role, default_password_lifetime, default_storage_engine</p>
<p>Database Instance-Level parameters Every instance in an Amazon Aurora cluster can be associated with a unique database parameter group</p>	<p>Managed using database parameter groups. For example: autocommit, connect_timeout, innodb_change_buffer_max_size</p>

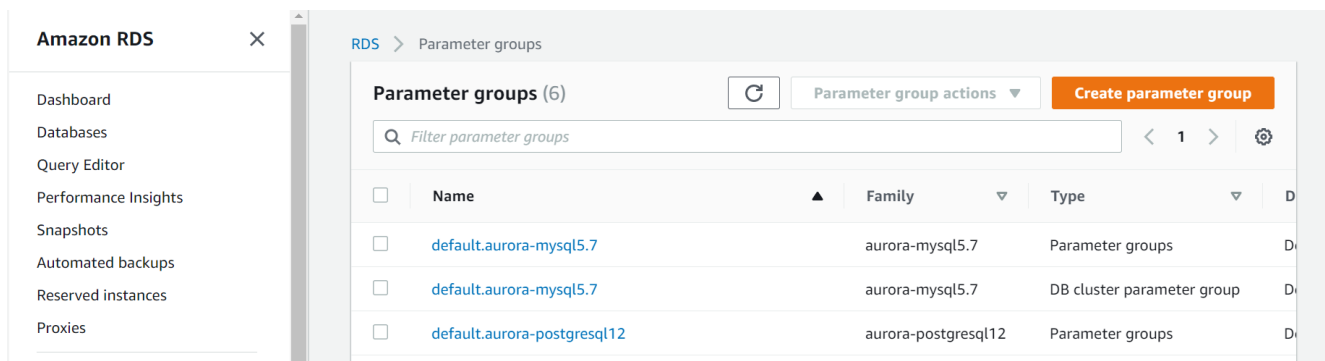
Examples

Create and Configure a New Parameter Group

Follow the steps below to create and configure Amazon Aurora database and cluster parameter groups:

1. Navigate to the "[Parameter groups](#)" section in the RDS Service of the AWS Console.
2. Click **Create Parameter Group**.

Note: You cannot edit the default parameter group. You must create a custom parameter group to apply changes to your Amazon Aurora cluster and its database instances.



3. Select **aurora-mysql5.7** from the **Parameter group family** drop-down list. Select **DB Parameter Group** from the **Type** drop-down list (another option is to select **Cluster Parameter Group** for modifying cluster parameters). Click **Create**.

Create parameter group

Parameter group details
To create a parameter group, choose a parameter group family, then name and describe your parameter group

1 Parameter group family
DB family that this DB parameter group will apply to

aurora-mysql5.7

Type

DB Parameter Group

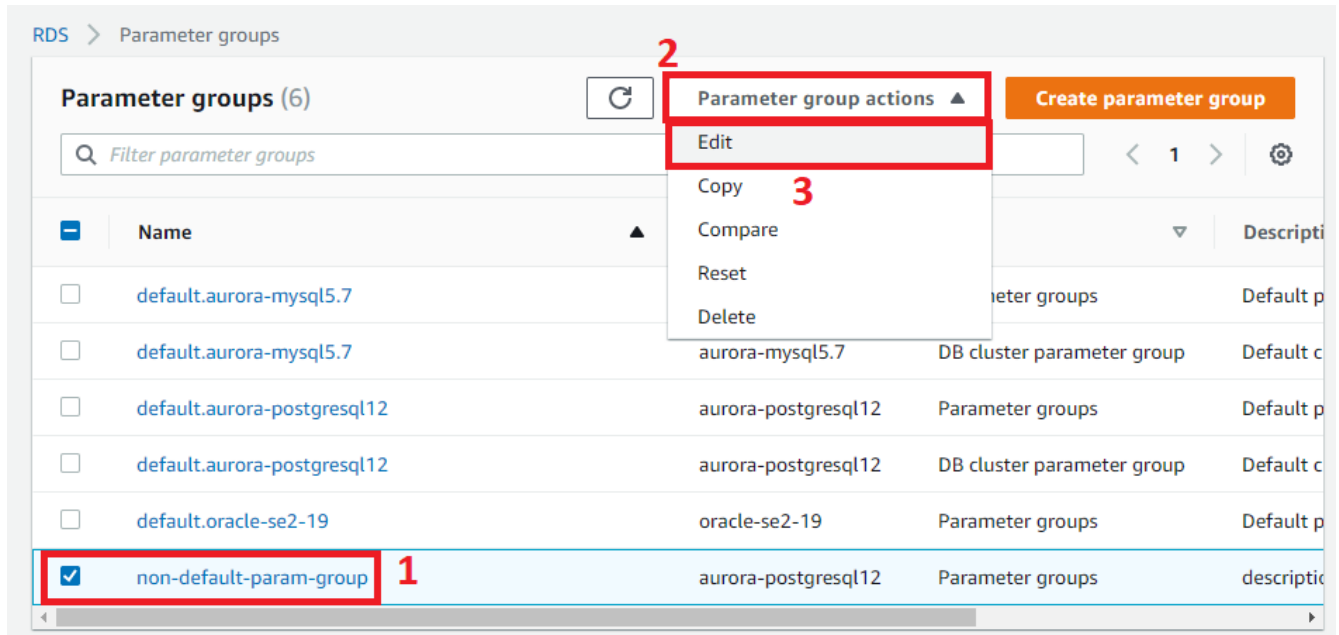
2 Group name
Identifier for the DB parameter group

Description
Description for the DB parameter group

Cancel **3** **Create**

Modify an Existing Parameter Group


1. Navigate to the "[Parameter groups](#)" section in the RDS Service of the AWS Console.
2. Click the name of the parameter to edit.
3. Click the **Edit parameters** button.



4. Change parameter values and click **Save changes**.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html>

Oracle Session Parameters vs. MySQL Session Variables

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	SET options are significantly different

Oracle Usage

Certain Oracle database parameters and configuration options are modifiable at the session level using the ALTER SESSION command. However, not all Oracle configuration options and parameters can be modified on a per-session basis. To view a list of all configurable parameters that can be set for the scope of a specific session, query the v\$parameter view as shown in the following example.

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE ISSES_MODIFIABLE='TRUE';
```

Examples

Change the NLS_LANGUAGE (codepage) parameter of the current session.

```
alter session set nls_language='SPANISH'

Sesió n modificada.

alter session set nls_language='ENGLISH';

Session altered.

alter session set nls_language='FRENCH';

Session modifi@e.

alter session set nls_language='GERMAN';

Session wurde geÄndert.
```

Specify the format of date values returned from the database using the NLS_DATE_FORMAT session parameter.

```
select sysdate from dual;

SYSDATE
-----
SEP-09-17

alter session set nls_date_format='DD-MON-RR';
Session altered.
```

```

select sysdate from dual;
SYSDATE
-----
09-SEP-17

alter session set nls_date_format='MM-DD-YYYY';

Session altered.

select sysdate from dual;

SYSDATE
-----
09-09-2017

alter session set nls_date_format='DAY-MON-RR';

Session altered.

```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/changing-parameter-values-in-a-parameter-file.html#GUID-4C578B21-DE2B-4210-8EB7-EF28D36CC1CB>

MySQL Usage

MySQL provides session-modifiable parameters configured using the SET SESSION command. Configuration of parameters using SET SESSION is only applicable in the current session. To view the list of parameters that can be set with SET SESSION, see <https://dev.mysql.com/doc/refman/5.7/en/dynamic-system-variables.html> and search for variables with session scope.

Examples of commonly used session parameters:

- **autocommit**: Specify if changes take effect immediately or if an explicit COMMIT command is required.
- **character_set_client**: Set the character set for the client.
- **default_storage_engine**: Set the default storage engine.
- **foreign_key_checks**: Set whether or not to run FK checks.
- **innodb_lock_wait_timeout**: Set how much time the transaction should wait to acquire a row lock.

Examples

Change the Time zone of the connected session.

```

SELECT now();

now()
-----|
2018-02-26 12:13:25 |

SET SESSION TIME_ZONE = '+10:00';
SELECT now();

```

```
now () |
-----|
2018-02-26 22:14:03 |
```

You can also use a time zone name such as 'Europe/Helsinki' instead of '+10:00'.

Oracle vs. MySQL Session parameter examples

Parameter purpose	Oracle	MySQL
Configure time and date format	ALTER SESSION SET nls_date_format = 'dd/m- m/yyyy hh24:mi:ss';	N/A
Configure the current default schema/database	ALTER SESSION SET current schema='schema_ name'	N/A
Generate traces for specific errors	ALTER SESSION SET events '10053 trace name context forever';	N/A
Run trace for a SQL statement	ALTER SESSION SET sql_trace=TRUE; ALTER SYSTEM SET EVENTS 'sql_trace [sql:&&sql_id] bind=true, wait=true';	SET GLOBAL general_log = 'ON';
Modify query optimizer cost for index access	ALTER SESSION SET optimizer_index_cost_adj = 50	SET SESSION optimizer_switch= ? You can turn on and off other strategies. See https://dev.mysql.com/doc/refman/5.7/en/switchable-optimizations.html
Modify query optimizer row access strategy	ALTER SESSION SET optimizer_mode=all_rows;	SET SESSION optimizer_switch= ? You can turn on and off other strategies. See https://dev.mysql.com/doc/refman/5.7/en/switchable-optimizations.html
Memory allocated to sort operations	ALTER SESSION SET sort_area_size=6321;	SET SESSION sort_buffer_size=32768;
Memory allocated to hash-joins	ALTER SESSION SET hash_area_size= e=1048576000;	SET SESSION join_buffer_size=1048576000;

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/set-variable.html>

Performance Tuning

Oracle Database Hints vs. MySQL Database Hints

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Very limited set of hints in MySQL - Index hints and optimizer hints as comments Syntax differences

Oracle Usage

Oracle provides users with the ability to influence how the query optimizer behaves and the decisions made to generate query execution plans. Controlling the behavior of the database optimizer is performed using Database Hints. They can be defined as a directive operation to the optimizer and alter the decisions of how execution plans are generated.

Oracle supports over 60 different database hints, and each database hint can have 0 or more arguments. Database hints are divided into different categories such as optimizer hints, join order hints, and parallel execution hints.

Database hints are embedded directly into the SQL queries immediately following the SELECT keyword using the format `/* <DB_HINT> */`.

Examples

Force the Query Optimizer to use a specific index for data access.

```

SELECT /* INDEX(EMP, IDX_EMP_HIRE_DATE) */ * FROM EMPLOYEES EMP
WHERE HIRE_DATE >= '01-JAN-2010';

Execution Plan
-----
Plan hash value: 3035503638
-----
-----
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
Time | | | | | | |
-----
-----
| 0 | SELECT STATEMENT | | 1 | 62 | 2 (0) |
00:00:01 |
| 1 | TABLE ACCESS BY INDEX ROWID | EMPLOYEES | 1 | 62 | 2 (0) |
00:00:01 |
|* 2 | INDEX RANGE SCAN | IDX_HIRE_DATE | 1 | | 1 (0) |
00:00:01 |
-----
-----
Predicate Information (identified by operation id):
-----
2 - access("HIRE_DATE">=TO_DATE(' 2010-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

```

For more details, see:

- <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Comments.html#GUID-D316D545-89E2-4D54-977F-FC97815CD62E>
- <https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/influencing-the-optimizer.html#GUID-8758EF88-1CC6-41BD-8581-246702414D1D>

MySQL Usage

Aurora MySQL supports two types of hints: Optimizer Hints and Index Hints.

Index Hints

The USE INDEX hint limits the optimizer's choice to one of the indexes listed in the <Index List> white list. Alternatively, indexes can be black listed using the IGNORE keyword.

The FORCE INDEX hint is similar to USE INDEX (index_list), but with strong favor towards seek vs. scan.

The hints use the actual index names, not column names. You can refer to primary keys using the keyword PRIMARY.

Syntax

```
SELECT ...
FROM <Table Name>
    USE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (<Index List>)
| IGNORE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (<Index List>)
| FORCE {INDEX|KEY}
        [FOR {JOIN|ORDER BY|GROUP BY}] (<Index List>)
...n
```

Note: In Aurora MySQL, the primary key is the clustered index.

The syntax for index hints has the following characteristics:

- Omitting <Index List> is allowed for USE INDEX only. It translates to "don't use any indexes", which is equivalent to a clustered index scan.
- Index hints can be further scoped down using the FOR clause. Use FOR JOIN, FOR ORDER BY or FOR GROUP BY to limit the hint applicability to that specific query processing phase.
- Multiple index hints can be specified for the same or different scope.

Optimizer Hints

Optimizer hints give developers or administrators control over some of the optimizer decision tree. They are specified within the statement text as a comment with the prefix "+".

Optimizer hints may pertain to different scopes and are valid in only one or two scopes. The available scopes for optimizer hints in descending scope width order are:

- **Global** hints affect the entire statement. Only MAX_EXECUTION TIME is a Global Optimizer Hint.
- **Query** level hints affect a query block within a composed statement such as UNION or a subquery.
- **Table** level hints affect a table within a query block.
- **Index** level hints affect an index of a table.

Syntax

```
SELECT /*+ <Optimizer Hints> */ <Select List>...
```

```
INSERT /*+ <Optimizer Hints> */ INTO <Table>...
```

```
REPLACE /*+ <Optimizer Hints> */ INTO <Table>...
```

```
UPDATE /*+ <Optimizer Hints> */ <Table> SET...
```

```
DELETE /*+ <Optimizer Hints> */ FROM <Table>...
```

The following optimizer hints are available in Aurora MySQL:

Hint Name	Description	Applicable Scopes
BKA, NO_BKA	Enables or disables Batched Key Access join processing	Query block, table
BNL ,NO_BNL	Enables or disables Block Nested-Loop join processing	Query block, table
MAX_EXECUTION_TIME	Limits statement execution time	Global
MRR, NO_MRR	Enables or disables Multi-Range Read optimization	Table, index
NO_ICP	Disables Index Condition Push-down optimization	Table, index
NO_RANGE_OPTIMIZATION	Disables range optimization	Table, index
QB_NAME	Assigns a logical name to a query block	Query block
SEMIJOIN, NO_SEMIJOIN	Enables or disables semi-join strategies	Query block
SUBQUERY	Determines MATERIALIZATION, and INTOEXISTS processing	Query block

Query block names (using QB_NAME) are used to distinguish a block for limiting the scope of the table hint. Add "@" to indicate a hint scope for one or more named subqueries. For example:

```
SELECT /*+ SEMIJOIN(@SubQuery1 FIRSTMATCH, LOOESCAN) */ *
FROM Table1
WHERE Col1 IN (SELECT /*+ QB_NAME(SubQuery1) */ Col1
              FROM t3);
```

Values for MAX_EXECUTION_TIME are measured in seconds and are always global for the entire query.

Note: This option does not exist in Oracle, where the execution time limit pertains to the session scope.

Migration Considerations

In general, the Aurora MySQL hint framework is relatively limited compared to the granular control provided by Oracle. It is recommended to start migration testing with all hints removed. Then, selectively apply hints as a last resort if other means such as schema, index, and query optimizations have failed.

Aurora MySQL uses a list of indexes and hints, both white list (USE) and black list (IGNORE), as opposed to Oracle's explicit index approach.

Index hints are not mandatory instructions. Aurora MySQL may choose alternatives if it cannot use the hinted index.

Examples

Force an index access.

```
SELECT * FROM Table1 USE INDEX (Index1) ORDER BY Col1;
```

Specify multiple index hints.

```
SELECT * FROM Table1 USE INDEX (Index1) INNER JOIN Table2 IGNORE INDEX(Index2) ON
Table1.Col1 = Table2.Col1 ORDER BY Col1;
```

Specify optimizer hints.

```
SELECT /*+ NO_RANGE_OPTIMIZATION(Table1 PRIMARY, Index2) */ Col1 FROM Table1 WHERE
Col2 = 300;
```

```
SELECT /*+ BKA(t1) NO_BKA(t2) */ * FROM Table1 INNER JOIN Table2 ON ...;
```

```
SELECT /*+ NO_ICP(t1, t2) */ * FROM Table1 INNER JOIN Table2 ON ...;
```


Summary

Feature	Oracle	Aurora MySQL
Force a specific plan	DBMS_SPM	N/A
Join hints	USE_NL NO_USE_NL USE_NL_ WITH_INDEX USE_MERGE NO_USE_MERGE USE_HASH NO_USE_HASH	BNL NO_BNL (Block Nested Loops)
Force scan	FULL	USE with no index list forces a clustered index scan
Force an index	INDEX	USE
White list and black list indexes	NO_INDEX	Supported with USE and IGNORE
Parameter value hints	opt_param	N/A

For more information, see:

- <https://dev.mysql.com/doc/refman/5.7/en/controlling-optimizer.html>
- <https://dev.mysql.com/doc/refman/5.7/en/optimizer-hints.html>
- <https://dev.mysql.com/doc/refman/5.7/en/index-hints.html>
- <https://dev.mysql.com/doc/refman/5.7/en/subquery-optimization.html>

Oracle Execution Plans vs. MySQL Execution Plans

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Syntax differences Completely different optimizer with different operators and rules

Oracle Usage

Execution plans represent the choices made by the query optimizer for accessing database data. The query optimizer generates execution plans for SELECT, INSERT, UPDATE and DELETE statements. Users and database administrators can view execution plans for specific queries and DML operations.

Execution plans are especially useful for performance tuning of queries. For example, determining if new indexes should be created. Execution plans can be affected by data volumes, data statistics, and instance parameters (i.e. Global or Session level parameters).

Execution plans are displayed as a structured tree with the following information:

- Tables access by the SQL statement and the referenced order for each table.
- Access method for each table in the statement (full table scan vs. index access).
- Algorithms used for join operations between tables (hash vs. nested loop joins).
- Operations performed on retrieved data as such as filtering, sorting, and aggregations.
- Information about rows being processed (cardinality) and the cost for each operation.
- Table partitions being accessed.
- Information about parallel executions.

Oracle 19 introduces SQL Quarantine: now queries that consume resources excessively can be automatically quarantined and prevented from being executed. These queries execution plans are also quarantined.

Examples

Review the potential execution plan for a query using the EXPLAIN PLAN statement.

```

SET AUTOTRACE TRACEONLY EXPLAIN
SELECT EMPLOYEE_ID, LAST_NAME, FIRST_NAME FROM EMPLOYEES
WHERE LAST_NAME='King' AND FIRST_NAME='Steven';

```

Execution Plan

```

-----
Plan hash value: 2077747057
-----
-----
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|-----|-----|-----|-----|-----|-----|-----|
-----
| 0 | SELECT STATEMENT | | 1 | 16 | 2 (0) | |

```

```

00:00:01 |
| 1 | TABLE ACCESS BY INDEX ROWID| EMPLOYEES | 1 | 16 | 2 (0) |
00:00:01 |
|* 2 | INDEX RANGE SCAN | EMP_NAME_IX | 1 | | 1 (0) |
00:00:01 |
-----
-----
Predicate Information (identified by operation id):
-----
2 - access("LAST_NAME"='King' AND "FIRST_NAME"='Steven')

```

SET AUTOTRACE TRACEONLY EXPLAIN instructs SQL*PLUS to show the execution plan without actually running the query itself.

The EMPLOYEES table contains indexes for both the LAST_NAME and FIRST_NAME columns. Step 2 of the execution plan above indicates the optimizer is performing an INDEX RANGE SCAN in order to retrieve the filtered employee name.

View a different execution plan displaying a FULL TABLE SCAN.

```

SET AUTOTRACE TRACEONLY EXPLAIN
SELECT EMPLOYEE_ID, LAST_NAME, FIRST_NAME FROM EMPLOYEES
WHERE SALARY > 10000;
Execution Plan
-----
Plan hash value: 1445457117
-----
| Id | Operation          | Name          | Rows | Bytes | Cost (%CPU)| Time     |
-----
| 0 | SELECT STATEMENT   |               |    72 | 1368 | 3 (0)| 00:00:01 |
|* 1 | TABLE ACCESS FULL| EMPLOYEES     |    72 | 1368 | 3 (0)| 00:00:01 |
-----
Predicate Information (identified by operation id):
-----
1 - filter("SALARY">10000)

```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/generating-and-displaying-execution-plans.html#GUID-60E30B1C-342B-4D71-B154-C26623D6A3B1>

MySQL Usage

Aurora MySQL provides the EXPLAIN/DESCRIBE statement—used with the SELECT, DELETE, INSERT, REPLACE, and UPDATE statements—to display execution plans.

Note: You can use the EXPLAIN/DESCRIBE statement to retrieve table and column metadata.

When EXPLAIN is used with a statement, MySQL returns the execution plan generated by the query optimizer. MySQL explains how the statement will be processed including information about table joins and order.

When EXPLAIN is used with the FOR CONNECTION option, it returns the execution plan for the statement executing in the named connection. You can use the FORMAT option to select either a TRADITIONAL tabular format or JSON.

The EXPLAIN statement requires SELECT permissions for all tables and views accessed by the query directly or indirectly. For views, EXPLAIN requires the SHOW VIEW permission.

EXPLAIN can be extremely valuable for improving query performance when used to find missing indexes. You can also use EXPLAIN to determine if the optimizer joins tables in an optimal order.

MySQL Workbench includes an easy to read visual explain feature similar to Oracle Execution Manager (OEM) graphical execution plans.

RDS MySQL Only (not Aurora MySQL): A new form of the EXPLAIN statement EXPLAIN ANALYZE is implemented in MySQL 8.0.18 providing expanded information about the execution of SELECT statements in TREE format for each iterator used in processing the query and making it possible to compare estimated cost with the actual cost of the query. This information includes start-up cost, total cost, number of rows returned by this iterator and the number of loops executed. In MySQL 8.0.21 and later, this statement also supports a FORMAT=TREE specifier. TREE is the only supported format. See [Obtaining Information with EXPLAIN ANALYZE](#) for more information.

Syntax

Simplified syntax for the EXPLAIN statement:

```
{EXPLAIN | DESCRIBE | DESC} [EXTENDED | FORMAT = TRADITIONAL | JSON]
[SELECT statement | DELETE statement | INSERT statement | REPLACE statement | UPDATE
statement | FOR CONNECTION <connection id>]
```

Examples

View the execution plan for a statement.

```
CREATE TABLE Employees (EmployeeID INT NOT NULL PRIMARY KEY, Name VARCHAR(100) NOT
NULL, INDEX USING BTREE(Name));
```

```
EXPLAIN SELECT * FROM Employees WHERE Name = 'Jason';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows
1	SIMPLE	Employees		ref	Name	Name	102	const	1

View the MySQL Workbench graphical execution plan.

MySQL Workbench interface showing a query and its execution plan. The query is:

```
SELECT * FROM orders
WHERE o_orderdate BETWEEN '1992-04-01' AND '1992-04-30'
AND o_clerk LIKE '%0223';
```

The execution plan shows an Index Range Scan on the orders table using the index i_o_orderdate. The plan details include:

- Access Type: range
- Index Range Scan
- Cost Hint: Medium - partial index scan
- Key/Index: i_o_orderdate
- Used Key Parts: o_orderDATE
- Possible Keys: i_o_orderdate
- Attached Condition: ('dbt3', 'orders', 'o_clerk' like '%0223')
- Rows Examined per Scan: 32642
- Rows Produced per Join: 32642
- Filtered (ratio of rows produced per rows examined): 100%
- Hint: 100% is best, <= 1% is worst


The bottom panel shows the output of the EXPLAIN query:

Time	Action	OK	0.000 sec
1 00:00:44	SELECT * FROM orders WHERE o_orderdate BETWEEN '1992-04-01' AND '1992-04-30' AND o_clerk LIKE '%0223' LIMIT 0, 1000	18 row(s) return...	0.281 sec / 0.000 sec
2 00:00:49	EXPLAIN SELECT * FROM orders WHERE o_orderdate BETWEEN '1992-04-01' AND '1992-04-30' AND o_clerk LIKE '%0223'	OK	0.000 sec

Note: To instruct the optimizer to use a join order corresponding to the order in which the tables are specified in a SELECT statement, use SELECT STRAIGHT_JOIN.

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/explain.html>

Oracle Table Statistics vs. MySQL Managing Statistics

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Syntax and option differences, similar functionality

Oracle Usage

Table statistics are one of the important aspects affecting SQL query performance. They enable the query optimizer to make informed assumptions when deciding how to generate the execution plan for each query. Oracle provides the DBMS_STATS package to manage and control the table statistics, which can be collected automatically or manually.

The following statistics are usually collected on database tables and indexes:

- Number of table rows.
- Number of table blocks.
- Number of distinct values or nulls.
- Data distribution histograms.

Automatic Optimizer Statistics Collection

By default, Oracle collects table and index statistics during predefined maintenance windows using the database scheduler and automated maintenance tasks. The automatic statistics collection mechanism uses Oracle's data modification monitoring feature that tracks the approximate number of INSERTs, UPDATEs, and DELETEs to determine which table statistics should be collected.

Oracle 19 now allows to gather real-time statistics on tables during regular UPDATE, INSERT and DELETE operations, which ensures that statistics are always up-to-date and are not going stale.

Oracle 19 also introduces High-Frequency Automatic Optimizer Statistics Collection - ability to set up automatic task that will collect statistics for stale objects.

Manual Optimizer Statistics Collection

When the automatic statistics collection is not suitable for a particular use case, the optimizer statistics collection can be performed manually at several levels:

Statistics level	Description
GATHER_INDEX_STATS	Index statistics
GATHER_TABLE_STATS	Table, column, and index statistics
GATHER_SCHEMA_STATS	Statistics for all objects in a schema
GATHER_DICTIONARY_STATS	Statistics for all dictionary objects
GATHER_DATABASE_STATS	Statistics for all objects in a database

Examples

Collect statistics at the table level (schema - HR, table - EMPLOYEES).

```
BEGIN
DBMS_STATS.GATHER_TABLE_STATS ('HR', 'EMPLOYEES');
END;
/

PL/SQL procedure successfully completed.
```

Collect statistics at a specific column level (schema - HR, table - EMPLOYEES, column - DEPARTMENT_ID).

```
BEGIN
DBMS_STATS.GATHER_TABLE_STATS ('HR', 'EMPLOYEES',
METHOD_OPT=>'FOR COLUMNS department_id');
END;
/

PL/SQL procedure successfully completed.
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/optimizer-statistics-concepts.html#GUID-C0E74ACE-2706-48A1-97A2-33F52207166A>

MySQL Usage

Aurora MySQL supports two modes of statistics management: Persistent Optimizer Statistics and Non-Persistent Optimizer Statistics. As the name suggests, persistent statistics are written to disk and survive service restart. Non-persistent statistics are kept in memory and need to be recreated after service restart. It is recommended to use persistent optimizer statistics (the default for Aurora MySQL) for improved plan stability.

Statistics in Aurora MySQL are created for indexes only. Aurora MySQL does not support independent statistics objects on columns that are not part of an index.

Typically, administrators change the statistics management mode by setting the global parameter `innodb_stats_persistent = ON`. Therefore, control the statistics management mode by changing the behavior for individual tables using the table option `STATS_PERSISTENT = 1`. There are no column-level or statistics-level options for setting parameter values.

To view statistics metadata, use the `INFORMATION_SCHEMA.STATISTICS` standard view. To view detailed persistent optimizer statistics, use the `innodb_table_stats` and `innodb_index_stats` tables.

Example of `mysql.innodb_table_stats` content

database_name	table_name	last_update	n_rows	clustered_index_size	sum_of_other_index_sizes
mysql	aurora_s3_load_history	2021-06-16 01:44:29	0	1	0
mysql	columns_priv	2017-07-14 00:25:33	0	1	0
mysql	db	2017-07-14 00:25:33	0	1	1
mysql	event	2017-10-31 02:20:35	0	1	0
mysql	func	2017-07-14 00:25:33	0	1	0
mysql	host	2017-10-26 21:31:42	0	1	0
mysql	ndb_binlog_index	2017-07-13 18:08:32	0	1	0
mysql	proc	2021-06-16 01:44:40	79	97	0
mysql	procs_priv	2017-10-31 02:20:25	0	1	1
mysql	proxies_priv	2017-07-13 18:08:32	0	1	1
mysql	rds_configuration	2017-07-13 18:08:32	0	1	0
mysql	rds_global_status_history	2017-07-13 18:08:32	0	1	0
mysql	rds_global_status_history_old	2017-07-13 18:08:32	0	1	0
mysql	rds_heartbeat2	2017-07-13 18:08:32	0	1	0
mysql	rds_history	2021-06-16 01:44:30	1	1	0
mysql	rds_replication_status	2017-10-31 02:18:52	0	1	0
mysql	rds_sysinfo	2017-07-13 18:08:32	0	1	0

Example of mysql.innodb_index_stats content

database_name	table_name	index_name	last_update	stat_name	stat_value	sample_size	stat_description
mysql	aurora_s3_load_history	GEN_CLUST_INDEX	2021-06-16 01:44:29	n_diff_pfx01	0	1	DB_ROW_ID
mysql	aurora_s3_load_history	GEN_CLUST_INDEX	2021-06-16 01:44:29	n_leaf_pages	1	1	Number of leaf pages in the index
mysql	aurora_s3_load_history	GEN_CLUST_INDEX	2021-06-16 01:44:29	size	1	1	Number of pages in the index
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_diff_pfx01	0	1	Host
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_diff_pfx02	0	1	Host,Db
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_diff_pfx03	0	1	Host,Db,User
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_diff_pfx04	0	1	Host,Db,User,Table_name
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_diff_pfx05	0	1	Host,Db,User,Table_name,Column_name
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	n_leaf_pages	1	1	Number of leaf pages in the index
mysql	columns_priv	PRIMARY	2017-07-14 00:25:33	size	1	1	Number of pages in the index
mysql	db	PRIMARY	2017-07-14 00:25:33	n_diff_pfx01	0	1	Host
mysql	db	PRIMARY	2017-07-14 00:25:33	n_diff_pfx02	0	1	Host,Db
mysql	db	PRIMARY	2017-07-14 00:25:33	n_diff_pfx03	0	1	Host,Db,User
mysql	db	PRIMARY	2017-07-14 00:25:33	n_leaf_pages	1	1	Number of leaf pages in the index
mysql	db	PRIMARY	2017-07-14 00:25:33	size	1	1	Number of pages in the index
mysql	db	User	2017-07-14 00:25:33	n_diff_pfx01	0	1	User
mysql	db	User	2017-07-14 00:25:33	n_diff_pfx02	0	1	User,Host
mysql	db	User	2017-07-14 00:25:33	n_diff_pfx03	0	1	User,Host,Db
mysql	db	User	2017-07-14 00:25:33	n_leaf_pages	1	1	Number of leaf pages in the index
mysql	db	User	2017-07-14 00:25:33	size	1	1	Number of pages in the index
mysql	event	PRIMARY	2017-10-31 02:20:35	n_diff_pfx01	0	1	db

Automatic refresh of statistics is controlled by the global parameter `innodb_stats_auto_recalc`, which is set to ON in Aurora MySQL. You can set it individually for each table using the `STATS_AUTO_RECALC=1` option.

To explicitly force a refresh of table statistics, use the `ANALYZE TABLE` statement. It is not possible to refresh individual statistics or columns.

Use the `NO_WRITE_TO_BINLOG`, or its clearer alias `LOCAL`, to avoid replication to replication slaves.

Use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more individual partitions.

RDS MySQL Only (not Aurora MySQL): MySQL 8 adds new `INFORMATION_SCHEMA.INNODB_CACHED_INDEXES` table which reports the number of index pages cached in the InnoDB buffer pool for each index.

Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE <Table Name> [, ...];
```

```
CREATE TABLE ( <Table Definition> ) | ALTER TABLE <Table Name>
STATS_PERSISTENT = <1|0>,
STATS_AUTO_RECALC = <1|0>,
STATS_SAMPLE_PAGES = <Statistics Sampling Size>;
```

Migration Considerations

Unlike Oracle, Aurora MySQL collects only density information. It does not collect detailed key distribution histograms. This difference is critical for understanding execution plans and troubleshooting performance issues that are not affected by individual values used by query parameters.

Statistics collection is managed at the table level. You cannot manage individual statistics objects or individual columns. In most cases, that should not pose a challenge for successful migration.

Examples

Create a table with explicitly set statistics options.

```
CREATE TABLE MyTable
(Col1 INT NOT NULL AUTO_INCREMENT,
Col2 VARCHAR(255),
DateCol DATETIME,
PRIMARY KEY (Col1),
INDEX IDX_DATE (DateCol)
) ENGINE=InnoDB,
STATS_PERSISTENT=1,
STATS_AUTO_RECALC=1,
STATS_SAMPLE_PAGES=25;
```

Refresh all statistics for MyTable1 and MyTable2.

```
ANALYZE TABLE MyTable1, MyTable2;
```

Change MyTable to use non-persistent statistics.

```
ALTER TABLE MyTable STATS_PERSISTENT=0;
```

Summary

The following table identifies Aurora MySQL features. All of the features are accessed in Oracle using the DBMS_STATS package.

Feature	Aurora MySQL	Comments
Column statistics	N/A	
Index statistics	Implicit with every index	Statistics are maintained automatically for every table index.
Refresh / update statistics	ANALYZE TABLE	Minimal scope in Aurora MySQL is the entire table. No control over individual statistics.
Auto create statistics	N/A	
Auto update statistics	STATS_AUTO_RECALC table option	
Statistics sampling	STATS_SAMPLE_PAGES table option	Can only use page number, not percentage for STATS_SAMPLE_PAGES.

Feature	Aurora MySQL	Comments
Full scan refresh	N/A	Using a very large STATS_SAMPLE_PAGES may server the same purpose.
Non-persistent statistics	Use STATS_PERSISTENT=0 table option	


For more information, see:

- <https://dev.mysql.com/doc/refman/5.7/en/statistics-table.html>
- <https://dev.mysql.com/doc/refman/5.7/en/innodb-persistent-stats.html>
- <https://dev.mysql.com/doc/refman/5.7/en/innodb-statistics-estimation.html>
- <https://dev.mysql.com/doc/refman/5.7/en/innodb-performance-optimizer-statistics.html>

Security



Oracle Encrypted Connections vs. Aurora for MySQL Encrypted Connections

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	

Oracle Usage

Oracle Database supports encrypting incoming data out of the box using native Oracle Net Services. You can encode data that is sent to and from the server using Advanced Encryption Standard (AES) algorithm, ARIA (Academia, Research Institute, and Agency) algorithm, GOSudarstvennyy Standard (GOST) algorithm, Korea Information Security Agency SEED algorithm and Triple-DES encryption (3DES).

Algorithms can be specified in sqlnet.ora file for the clients and servers.

For more information see [Configuring Oracle Database Network Encryption and Data Integrity](#)

SSL/TLS connections to the Oracle database are supported starting with 12c in the standard edition.

For more information see [SSL Connection to Oracle DB using JDBC, TLSv1.2, JKS or Oracle Wallets \(12.2 and lower\)](#)


MySQL Usage

MySQL supports encrypted connections between clients and the server using the TLS (Transport Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer) but MySQL does not actually use the SSL protocol for encrypted connections because its encryption is weak.

OpenSSL 1.1.1 supports the TLS v1.3 protocol for encrypted connections

RDS MySQL Only (not Aurora MySQL): MySQL 8.0.16 and higher supports TLS v1.3 as well if both the server and client are compiled using OpenSSL 1.1.1 or higher. See [Section 6.3.2, "Encrypted Connection TLS Protocols and Ciphers"](#)

Oracle Transparent Data Encryption (TDE) vs. MySQL AWS Aurora Encryption and Column Encryption

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Use AWS Aurora Encryption

Oracle Usage

Oracle uses Transparent Data Encryption (TDE) to encrypt data stored on media in order to provide "data at rest" protection. Although Oracle uses authentication, authorization, and auditing to secure data in the database, TDE provides additional security at the operating system level.

As the name implies, encryption operations are performed automatically and are transparent to client applications. However, TDE does not address data in transit, which must be handled by network security protocols.

Characteristics of TDE include:

- The ADMINISTER KEY MANAGEMENT system privilege is required to configure TDE.
- Data can be encrypted at the column level or the tablespace level.
- Key encryption is managed in the external TDE Master Encryption Module.
- There is one master key for each database.

Examples

Configure the Master Encryption Key

Specify the location of the encryption wallet using the ENCRYPTION_WALLET_LOCATION parameter. The options are:

- Regular Filesystem
- Multiple databases share the same file
- ASM file system
- ASM disk group

Register the key file in the ASM diskgroup.

```
ENCRYPTION_WALLET_LOCATION=
(SOURCE=
(METHOD=FILE)
(METHOD_DATA=
(DIRECTORY=+ASM_file_path_of_the_diskgroup)))
```

Create a Software Keystore

There are three types of software keystores:

- Password-based
- Auto-login
- Local auto-login

Create a password-based software keystore. The user must have ADMINISTER KEY MANAGEMENT or SYSKM privilege.

```
sqlplus c##sec_admin as syskm
Enter password: password
Connected.

ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY
password;

keystore altered.
```

Open a Keystore

When using a password-based keystore, it must be opened before creating TDE master encryption keys or accessing the keystore. Keystores are automatically opened when using auto-login or local auto login.

```
sqlplus c##sec_admin as syskm
Enter password: password
Connected.

ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY password;

keystore altered.
```

Set the Software Master Encryption Key

The master encryption key protects the TDE table and tablespace encryption keys. By default, the master encryption key is generated by TDE. To set the master encryption key, ensure the database is open in READ WRITE mode, connect with a user account having the required privileges (see above), and create the master key.

```
sqlplus c##sec_admin as syskm
Enter password: password
Connected.

ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY keystore_password WITH BACKUP USING
'emp_key_backup';

keystore altered.
```

Encrypt the Data

Create an encrypted column.

```
CREATE TABLE employee (
  FIRST_NAME VARCHAR2(128),
  LAST_NAME VARCHAR2(128),
  EMP_ID NUMBER,
  SALARY NUMBER(6) ENCRYPT);
```

Column data types support for encryption include BINARY_DOUBLE, BINARY_FLOAT, CHAR, DATE, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, NCHAR, NUMBER, NVARCHAR2, RAW (legacy or extended), TIMESTAMP (includes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE), VARCHAR2 (legacy or extended).

Column encryption cannot be used with the following features:

- Index types other than B-tree
- Range scan search through an index
- Synchronous change data capture
- Transportable tablespaces
- Columns used in foreign key constraints

You can change the encryption algorithm using the NO SALT clause to encrypt without an algorithm or the USING clause to specify an algorithm.

```
CREATE TABLE EMPLOYEE (
  FIRST_NAME VARCHAR2(128),
  LAST_NAME VARCHAR2(128),
  EMP_ID NUMBER ENCRYPT NO SALT,
  SALARY NUMBER(6) ENCRYPT USING '3DES168');
```

Change the algorithm on an existing table.

```
ALTER TABLE EMPLOYEE REKEY USING 'SHA-1';
```

Remove column encryption.

```
ALTER TABLE employee MODIFY (SALARY DECRYPT);
```

- Ensure a COMPATIBLE initialization parameter is set to 11.2.0.0 (minimum).
- Login to the database.
- Create the tablespace (you cannot modify an existing tablespace; you can only create new one). In this example, the first tablespace is created with AES256 algorithm and the second is created with the default algorithm.

```
sqlplus sec_admin@hrpdb
Enter password: password
Connected.

CREATE TABLESPACE encrypt_ts
DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf' SIZE 1M
ENCRYPTION USING 'AES256'
DEFAULT STORAGE (ENCRYPT);
CREATE TABLESPACE securespace_2
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
```

```
ENCRYPTION  
DEFAULT STORAGE (ENCRYPT) ;
```

For more information, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/asoag/introduction-to-transparent-data-encryption.html#GUID-62AA9447-FDCD-4A4C-B563-32DE04D55952>

MySQL Usage

Amazon provides the ability to encrypt “data at rest” (data stored in persistent storage). When data encryption is enabled, it automatically encrypts the database server storage, automated backups, read replicas, and snapshots using the AES-256 encryption algorithm. AWS KMS performs the encryption. For more information, see [What is AWS Key Management Service?](#)

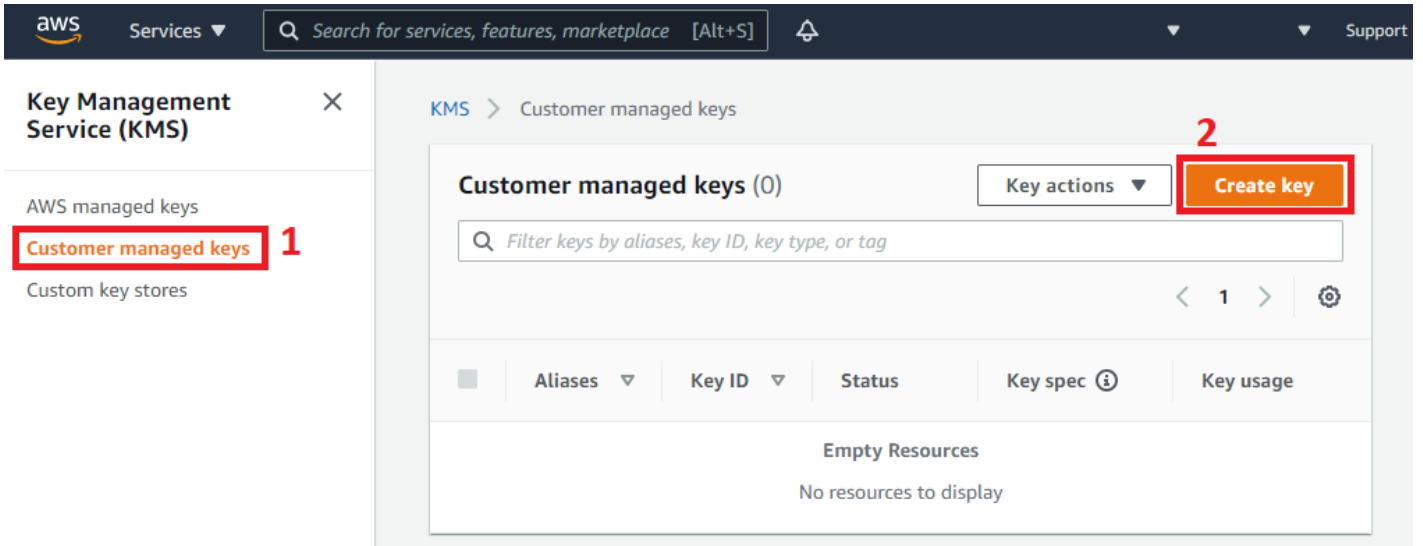
Once enabled, AWS transparently encrypts and decrypts the data without any impact on performance or any user intervention. There is no need to modify clients to support encryption.

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports FIPS mode if compiled using OpenSSL and an OpenSSL library and FIPS Object Module are available at runtime. FIPS mode imposes conditions on cryptographic operations such as restrictions on acceptable encryption algorithms or requirements for longer key lengths. See [Section 6.8 FIPS Support](#).

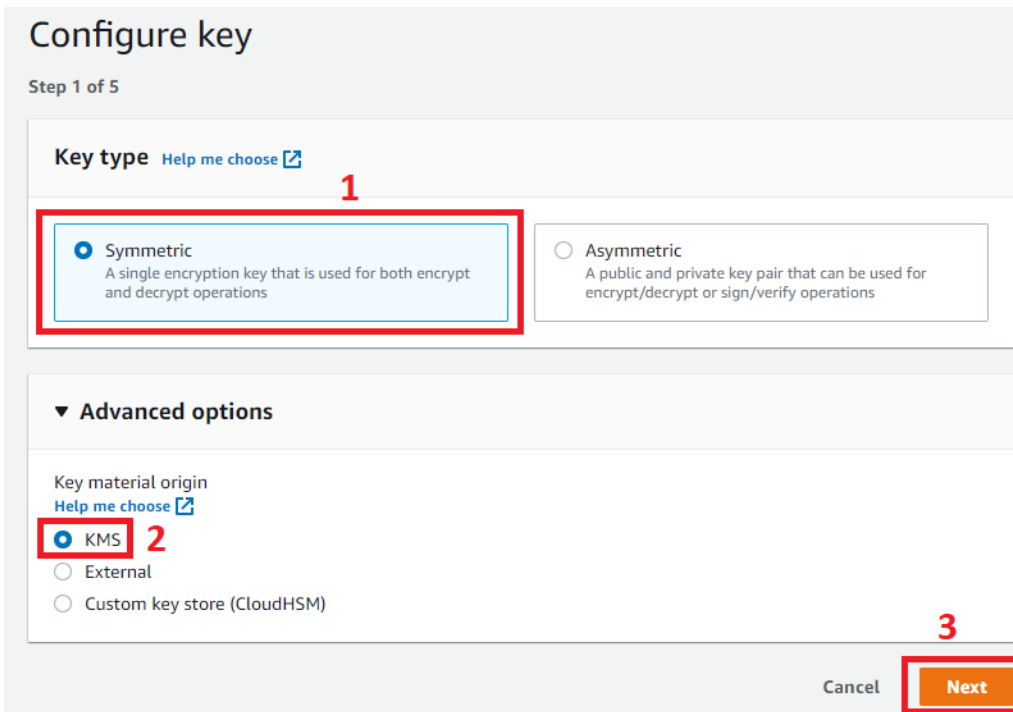
Table encryption can now be managed globally by defining and enforcing encryption defaults. The `default_table_encryption` variable defines an encryption default for newly created schemas and general tablespaces. The encryption default for a schema can also be defined using the `DEFAULT ENCRYPTION` clause when creating a schema. By default a table inherits the encryption of the schema or general tablespace it is created in. Encryption defaults are enforced by enabling the `table_encryption_privilege_check` variable. The privilege check occurs when creating or altering a schema or general tablespace with an encryption setting that differs from the `default_table_encryption` setting or when creating or altering a table with an encryption setting that differs from the default schema encryption. The `TABLE_ENCRYPTION_ADMIN` privilege permits overriding default encryption settings when `table_encryption_privilege_check` is enabled. For more information see [Defining an Encryption Default for Schemas and General Tablespaces](#).

Creating an Encryption Key

To create your own key, browse to the Key Management Service (KMS) and click on "Customer managed keys" and create a new key.



Choose relevant options and click on “Next”.



Define Alias as the name of the key and click "Next".

Add labels

Alias
You can change the alias at any time. [Learn more](#)

1

Alias
db-finance

Description - optional
You can change the description at any time.

Description - optional
Description of the key

Tags - optional

You can use tags to categorize and identify your CMKs and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags.

2

Cancel Previous **Next**

You can skip “Define Key Administrative Permissions” (step 3 of 5), click on “Next”.

On the next step make sure to assign the key to the relevant users who will need to interact with Aurora:

Define key usage permissions

Step 4 of 5

This account
Select the IAM users and roles that can use the CMK in cryptographic operations. [Learn more](#)

Q

<input type="checkbox"/>	Name	Path
<input checked="" type="checkbox"/>	g am.com	/

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

Add another AWS account

Cancel Previous **Next**

On the last step you will be able to see the ARN of the key and its account.

Review and edit key policy

```

1 {
2   "Id": "key-consolepolicy-3",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "Enable IAM User Permissions",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "arn:aws:iam::          :root"
10      },
11      "Action": "kms:*",
12      "Resource": "*"
13    },
14    {
15      "Sid": "Allow use of the key",

```

Cancel Previous Finish

Click on “Finish” and now this key will be listed in under customer managed keys.

Now you will be able to set Master encryption key by using the ARN of the key that you have created or picking it from the list.

Encryption

Encryption

Enable Encryption
Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console. [Learn More](#).

Disable Encryption

Master key [info](#) ARN

Enter a key ARN arn:aws:kms:us-east-1:270324613865:key/75786f1
e.g.:arn:aws:kms:<region>:<accountID>:key/<key-id>

Description	Account	KMS key ID
None	None	None

Proceed to finish and launch the instance.

Enabling Encryption

As part of the database settings, you will be prompted to enable encryption and select a master key.

Encryption for an Amazon RDS DB instance can be enabled only during the instance creation.

Encryption

Encryption

Enable Encryption
Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console. [Learn More](#).

Disable Encryption

Master key [info](#)

(default) aws/rds

Master key

This is the master key that will be used to protect the key used to encrypt this database volume. You can select from master keys in your account or type/paste the ARN of a key from a different account. You can create a new master encryption key by going to the Encryption Keys tab of the IAM console.

You can select the default key provided for the account or define a specific key based on an IAM KMS ARN from your account or a different account.

SSE-S3 Encryption Feature Overview

Server-Side Encryption with Amazon S3-managed encryption keys (SSE-S3) uses multi-factor encryption. Amazon S3 encrypts its objects with a unique key and it also encrypts the key itself with a master key that rotates periodically.

SSE-S3 uses AES-256 as its encryption standard.

Once the S3 bucket is enabled with Server-side encryption, the data will be encrypted at rest. All API calls must now include the special header `-x-amz-server-side-encryption`.

For more information, see:

- SSE-23 with API - [Specifying Server-Side Encryption Using the REST API](#)
- SSE-S3 with AWS CLI S3 - [s3](#)

Follow these steps to enable SSE-S3:

1. Create an [AWS GLUE Job](#).
2. Define the role, bucket, and script and then open "Script libraries and job parameters (optional)".
3. Enable "Server-Side Encryption".
4. Submit and run the job.

From this point forward, the only way to access the files is to use AWS CLI s3 with the `--sse` switch, or by adding `x-amz-server-side-encryption` to your API calls.

Usage of Column Encryption

Aurora MySQL provides encryption and decryption functions similar to Oracle with a much less elaborate security hierarchy that is easier to manage.

The encryption functions require the actual key as a string, so you must take extra measures to protect the data. For example, hashing the key values on the client.

Aurora MySQL supports the AES and DES encryption algorithms. You can use the following functions for data encryption and decryption:

- AES_DECRYPT
- AES_ENCRYPT
- DES_DECRYPT
- DEC_ENCRYPT

Syntax

General syntax for the encryption functions:

```
[A|D]ES_ENCRYPT(<string to be encrypted>, <key string> [,<initialization vector>])
[A|D]ES_DECRYPT(<encrypted string>, <key string> [,<initialization vector>])
```

For more example, see: https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html#function_aes-encrypt

It is highly recommended to use the optional initialization vector to circumvent whole value replacement attacks. When encrypting column data, it is common to use an immutable key as the initialization vector. With this approach, decryption fails if a whole value moves to another row.

Consider using SHA2 instead of SHA1 or MD5 because there are known exploits available for the SHA1 and MD5. Passwords, keys, or any sensitive data passed to these functions from the client are not encrypted unless you are using an SSL connection. One benefit of using Amazon AWS IAM is that database connections are encrypted with SSL by default. See the [Users](#) and [Roles](#) sections for more information.

Examples

The following example demonstrates how to encrypt an employee Social Security Number:

Create an employees table.

```
CREATE TABLE Employees (EmployeeID INT NOT NULL PRIMARY KEY, SSN_Encrypted BINARY(32) NOT NULL);
```

Insert the encrypted data.

```
INSERT INTO Employees (EmployeeID, SSN_Encrypted)
VALUES (1, AES_ENCRYPT('1112223333', UNHEX(SHA2('MyPassword',512)), 1));
```

Note: Use the UNHEX function for more efficient storage and comparisons.


Verify decryption.

```
SELECT EmployeeID, SSN_Encrypted, AES_DECRYPT(SSN_Encrypted, UNHEX(SHA2('MyPassword',512)), EmployeeID) AS SSN
FROM Employees
```

EmployeeID	SSN_Encrypted	SSN
-----	-----	-----
1	` ©> +yp°øýNZ~Gø	1112223333

For more information, see <https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html>

Oracle Roles vs. MySQL Privileges

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	There are no roles - only privileges

Oracle Usage

Oracle roles are groups of privileges granted to database users. A database role can contain individual system and object permissions as well as other roles. Database roles enable you to grant multiple database privileges to users in one operation. It is convenient to group permissions together to ease the management of privileges.

Oracle 12c introduces a new multi-tenant database architecture that supports the creation of both common and local roles:

- **Common Roles:** Roles created at the container database (CDB) level. A common role is a database role that exists in the root and in every existing and future pluggable database (PDB). Common roles are useful for cross-container operations such as ensuring a common user has a role in every container.
- **Local Roles** Roles created in a specific pluggable database (PDB). A local role exists only in a single pluggable database and can only contain roles and privileges that apply within the pluggable database in which the role exists.

Notes:

- Common role names must start with a `c##` prefix. Beginning with Oracle 12.1.0.2, these prefixes can be changed using the `COMMON_USER_PREFIX` parameter.
- A `CONTAINER` clause can be added to `CREATE ROLE` statement to choose the container applicable for the role.

Example

Create a common role.

```
show con_name

CON_NAME
-----
CDB$ROOT

CREATE ROLE c##common_role;

Role created.
```

Create a local role.

```
show con_name

CON_NAME
-----
```

```
ORCLPDB

CREATE ROLE local_role;

Role created.
```

Grant privileges and roles to the local_role database role.

```
GRANT RESOURCE, ALTER SYSTEM, SELECT ANY DICTIONARY TO local_role;
```

Database users to which the local_role role is granted now have all privileges that were granted to the role.

Revoke privileges and roles from the local_role database role.

```
REVOKE RESOURCE, ALTER SYSTEM, SELECT ANY DICTIONARY FROM local_role;
```

For more details, see <https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/overview.html#GUID-2FBCFBBE-6B42-4DB8-83F3-55B63B75B1EB>

MySQL Usage

Currently in MySQL 5.7, there is no ROLE feature. You must specify required privileges. However, there is an option when granting privileges to use wild-card characters to specify multiple privileges on one or more objects.

RDS MySQL Only (not Aurora MySQL): MySQL 8 supports roles which are named collections of privileges. Roles can be created and dropped. Roles can have privileges granted to and revoked from them. Roles can be granted to and revoked from user accounts. The active applicable roles for an account can be selected from among those granted to the account and can be changed during sessions for that account.

For more information see [Using Roles](#).

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

RDS MySQL Only (not Aurora MySQL): MySQL 8 incorporates the concept of user account categories with system and regular users distinguished according to whether they have the SYSTEM_USER privilege.
See [Account Categories](#).

```
CREATE USER u1 IDENTIFIED BY 'password';

GRANT ALL ON *.* TO u1 WITH GRANT OPTION;

-- GRANT ALL includes SYSTEM_USER, so at this point
-- u1 can manipulate system or regular accounts
```


Examples

Grant privileges using a wild-card.

```
GRANT ALL ON test_db.* to 'testuser';  
GRANT CREATE USER on *.* to 'testuser';  
GRANT SELECT ON db2.* TO 'testuser';  
GRANT EXECUTE ON PROCEDURE mydb.myproc TO
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/grant.html>

Oracle Database Users vs. MySQL Users

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Syntax and option differences, similar functionality

Oracle Usage

Database user accounts are used for authenticating connecting sessions and authorizing access for individual users to specific database objects. Database Administrators grant privileges to user accounts, and applications use user accounts to access database objects.

Steps for Providing Database Access to Applications

1. Create a user account in the database. User accounts are typically authenticated using a password. Additional methods of authenticating users also exist.
2. Assign permissions to the database user account enabling access to certain database objects and system permissions.
3. Connecting applications authenticate using the database username and password.

Oracle Database Users Common Properties

- Granting privileges or roles (collection of privileges) to the database user.
- Defining the default database tablespace for the user.
- Assigning tablespace quotas for the user.
- Configuring password policy, password complexity, lock, or unlock the account.

Authentication Mechanisms

- **Username and Password:** (default)
- **External:** Using the operating system or third-party software (such as Kerberos).
- **Global:** Enterprise directory service (such as Active Directory or Oracle Internet Directory).

Oracle Schemas Compared to Users

In an Oracle database, a user equals a schema. This relationship is special because users and schemas are essentially the same thing. Consider an Oracle database user as the account you use to connect to a database while a database schema is the set of objects (tables, views, etc.) that belong to that account.

- You cannot create schemas and users separately. When you create a database user, you also create a database schema with the same name.
- When you run the CREATE USER command in Oracle, you create a user for login and a schema in which

to store database objects.

- Newly created schemas are empty, but objects such as tables can be created within them.

Database Users in Oracle 12c

Two types of users exist in the Oracle 12c database:

- **Common Users:** Created in all database containers, root, and Pluggable Databases (PDB). Common users must have the C## prefix in the username.
- **Local Users:** Created only in a specific PDB. Different database users with identical usernames can be created in multiple PDBs.

Examples

The example below demonstrates the following operations:

- Create a common database user using the default tablespace.
- Grant privileges and roles to the user.
- Assign a profile to the user, unlock the account, and force the user to change the password (PASSWORD EXPIRE).
- Create a local database user in the my_pdb1 pluggable database.

```
CREATE USER c##test_user IDENTIFIED BY password DEFAULT TABLESPACE USERS;
GRANT CREATE SESSION TO c##test_user;
GRANT RESOURCE TO c##test_user;
ALTER USER c##test_user ACCOUNT UNLOCK;
ALTER USER c##test_user PASSWORD EXPIRE;
ALTER USER c##test_user PROFILE ORA_STIG_PROFILE;
ALTER SESSION SET CONTAINER = my_pdb1;
CREATE USER app_user1 IDENTIFIED BY password DEFAULT TABLESPACE USERS;
```

For more details, see [Managing Security for Oracle Database Users](#)

MySQL Usage

Database user accounts are used for authenticating connecting sessions and authorizing access for individual users to specific database objects. Database Administrators grant privileges to database user accounts that are used by applications to authenticate with an Aurora MySQL database.

For each account, CREATE USER creates a new row in the mysql.user system table. The account row reflects the properties specified in the statement. Unspecified properties are set to their default values:

- **Authentication:** The authentication plugin defined by the default_authentication_plugin system variable, and empty credentials.
- **SSL/TLS:** NONE
- **Resource limits:** Unlimited
- **Password management:** PASSWORD EXPIRE DEFAULT
- **Account locking:** ACCOUNT UNLOCK

When first created, accounts have no privileges. To assign privileges, use the GRANT statement.

Steps for Providing Database Access to Applications

1. Create a user account in the database. Typically, users authenticate using a username and password. Additional methods of authenticating users also exist.
2. Assign permissions to the database user account enabling access to certain database objects and system permissions.
3. Connecting applications use the database username and password combination to authenticate with the database.

MySQL Database Users Common Properties

- Granting privileges to the database user.
- Configuring password policy, password complexity, lock, or unlock the account.
- Specifying authentication methods
- User naming to indicate from which host names the user can login.
- Profiling, for example: MAX_QUERIES_PER_HOUR or MAX_USER_CONNECTIONS

Authentication Mechanisms

- **Username and Password:** (default)
- **External:** Using the operating system or third-party software (IAM user).
- **Global:** Enterprise directory service (such as Active Directory)
- Etc.

IAM Authentication

This feature is the equivalent to Oracle OS authentication.

With Amazon RDS for MySQL or Aurora with MySQL compatibility, you can authenticate to your DB instance or DB cluster using AWS Identity and Access Management (IAM) database authentication. With this authentication method, you don't need to use a password when you connect to a DB instance. Instead, you use an authentication token.

IAM database authentication provides the following benefits:

- Network traffic to and from the database is encrypted using Secure Sockets Layer (SSL).
- You can use IAM to centrally manage access to your database resources, instead of managing access individually on each DB instance or DB cluster.
- For applications running on Amazon EC2, you can use EC2 instance profile credentials to access the database instead of a password, for greater security.

Note: With IAM database authentication, you are limited to a maximum of 20 new connections per second

Examples

The example below demonstrates the following operations:

- Create a database user using the password expired option (PASSWORD EXPIRE).
- Grant privileges to the user.
- Assign profiling properties to the user.

```
CREATE USER 'testuser'  
          IDENTIFIED BY 'new_password' PASSWORD EXPIRE;  
GRANT ALL ON test_db.* to 'testuser';  
GRANT CREATE USER on *.* to 'testuser';  
ALTER USER 'testuser' WITH MAX_QUERIES_PER_HOUR 90;
```

To create IAM user, make sure that IAM user or role exists and is named by the same database username

```
CREATE USER jane_doe IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```



For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/create-user.html>
- <https://dev.mysql.com/doc/refman/5.7/en/account-names.html>
- [IAM Database Authentication for MySQL and PostgreSQL](#)

Physical Storage



Oracle Table Partitioning vs. MySQL Table Partitioning

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		Partitions	Interval Partitioning, Partition Advisor, Preference Partitioning, Virtual Column Based Partitioning, and Automatic List Partitioning are not supported in Aurora MySQL

Oracle Usage

The purpose of database partitioning is to provide support for very large tables and indexes by splitting them into smaller pieces. Each partition has its own name and definitions. They can be managed separately or collectively as one object. From an application perspective, partitions are transparent. Partitioned tables behave the same as non-partitioned tables allowing your applications access using unmodified SQL statements. Table partitioning provides several benefits:

- **Performance Improvements:** Table partitions help improve query performance by accessing a subset of a partition instead of scanning a larger set of data. Additional performance improvements can be achieved when using partitions and parallel query execution for DML and DDL operations.
- **Data Management:** Table partitions facilitate easier data management operations (such as data migration), index management (creation, dropping, or rebuilding indexes), and backup/recovery. These operations are also referred to as Information Lifecycle Management (ILM) activities.
- **Maintenance Operations:** Table partitions can significantly reduce downtime caused by table maintenance operations.

Oracle 18c introduces following enhancements to partitioning:

- **Online Merging of Partitions and Subpartitions:** now it is possible to merge table partitions concurrently with Updates/Deletes and Inserts on a partitioned table.
- Oracle 18c also allows to modify partitioning strategy for the partitioned table: e.g. hash partitioning to range. This can be done both offline and online.

Oracle 19 introduces hybrid partitioned tables: partitions can now be both internal Oracle tables and external tables and sources. It is also possible to integrate both internal and external partitions together in a single partitioned table.

Hash Table Partitioning

When a partition key is specified (for example, a table column with a number data type), Oracle applies a hashing algorithm to evenly distribute the data (records) among all defined partitions. The partitions have approximately the same size.

For example, create a hash-partitioned table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    NUMBER NOT NULL,
   EVENT_DATE  DATE    NOT NULL,
   EVENT_STR   VARCHAR2(500),
```

```

ERROR_CODE  VARCHAR2(10)
PARTITION BY HASH (ERROR_CODE)
PARTITIONS 3
STORE IN (TB1, TB2, TB3);

```

List Table Partitioning

You can specify a list of discrete values for the table partitioning key in the description of each partition. This type of table partitioning enables control over partition organization using explicit values. For example, partition “events” by error code values.

For example, create a list-partitioned table.

```

CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    NUMBER NOT NULL,
   EVENT_DATE  DATE    NOT NULL,
   EVENT_STR   VARCHAR2(500),
   ERROR_CODE  VARCHAR2(10))
  PARTITION BY LIST (ERROR_CODE)
  (PARTITION warning VALUES ('err1', 'err2', 'err3') TABLESPACE TB1,
   PARTITION critical VALUES ('err4', 'err5', 'err6') TABLESPACE TB2);

```

Range Table Partitioning

Partition a table based on a range of values. The Oracle database assigns rows to table partitions based on column values falling within a given range. Range table partitioning is one of the most frequently used type of partitioning, primarily with date values. Range table partitioning can also be implemented with numeric ranges (1-10000, 10001- 20000...).

For example, create a range-partitioned table.

```

CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    NUMBER NOT NULL,
   EVENT_DATE  DATE    NOT NULL,
   EVENT_STR   VARCHAR2(500))
  PARTITION BY RANGE (EVENT_DATE)
  (PARTITION EVENT_DATE VALUES LESS THAN (TO_DATE('01/01/2015',
                                                    'DD/MM/YYYY')) TABLESPACE TB1,
   PARTITION EVENT_DATE VALUES LESS THAN (TO_DATE('01/01/2016',
                                                    'DD/MM/YYYY')) TABLESPACE TB2,
   PARTITION EVENT_DATE VALUES LESS THAN (TO_DATE('01/01/2017',
                                                    'DD/MM/YYYY')) TABLESPACE TB3);

```

Composite Table Partitioning

With composite partitioning, a table can be partitioned by one data distribution method, and then each partition can be further subdivided into subpartitions using the same, or different, data distribution method(s). For example:

- Composite list-range partitioning
- Composite list-list partitioning

- Composite range-hash partitioning

Partitioning Extensions

Oracle provides additional partitioning strategies that enhance the capabilities of basic partitioning. These partitioning strategies include:

- Manageability extensions
 - Interval partitioning
 - Partition advisor
- Partitioning key extensions
 - Reference partitioning
 - Virtual column-based partitioning

Split Partitions

The SPLIT PARTITION statement can be used to redistribute the contents of one partition, or sub-partition, into multiple partitions or sub-partitions.

```
ALTER TABLE SPLIT PARTITION p0 INTO
    (PARTITION p01 VALUES LESS THAN (100), PARTITION p02);
```

Exchange Partitions

The EXCHANGE PARTITION statement is useful to exchange table partitions in or out of a partitioned table.

```
ALTER TABLE orders EXCHANGE
    PARTITION p_ord3 WITH TABLE orders_year_2016;
```

Subpartitioning Tables

You can create Subpartitions within partitions to further split the parent partition.

```
PARTITION BY RANGE(department_id)
    SUBPARTITION BY HASH(last_name)
    SUBPARTITION TEMPLATE
        (SUBPARTITION a TABLESPACE ts1,
         SUBPARTITION b TABLESPACE ts2,
         SUBPARTITION c TABLESPACE ts3,
         SUBPARTITION d TABLESPACE ts4)
    (PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (2000),
     PARTITION p3 VALUES LESS THAN (MAXVALUE))
```

For more details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/19/vldbg/partition-concepts.html#GUID-EA7EF5CB-DD49-43AF-889A-F83AAC0D7D51>

Automatic List Partitioning (Oracle 12c only)

Automatic list partitioning is an enhancement that enables automatic creation of new partitions for new values inserted into a list-partitioned table. An automatic list-partitioned table is created with only one partition. The database creates the additional table partitions automatically.

For example, Create an automatic list-partitioned table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    NUMBER NOT NULL,
  EVENT_DATE  DATE    NOT NULL,
  EVENT_STR   VARCHAR2(500),
  ERROR_CODE  VARCHAR2(10))
  PARTITION BY LIST (ERROR_CODE) AUTOMATIC
  (PARTITION warning VALUES ('err1', 'err2', 'err3'))
```

For more details, see

<http://www.oracle.com/technetwork/database/options/partitioning/partitioning-wp-12c-1896137.pdf>

MySQL Usage

The Table Partitioning mechanism in MySQL is similar to Oracle and contains most of the Oracle Table Partitioning features. The only items not supported in MySQL Table Partitioning are the automatic features such as Interval Partitioning and Automatic List Partitioning. These features can be implemented using triggers or procedures. For more information, see <https://dev.mysql.com/doc/refman/5.7/en/partitioning.html>.

RDS MySQL Only (not Aurora MySQL): In MySQL 8 ADD PARTITION, DROP PARTITION, COALESCE PARTITION, REORGANIZE PARTITION and REBUILD PARTITION ALTER TABLE options are supported by native partitioning in-place APIs and may be used with ALGORITHM={COPY|INPLACE} and LOCK clauses. DROP PARTITION with ALGORITHM=INPLACE deletes data stored in the partition and drops the partition. However DROP PARTITION with ALGORITHM=COPY or old_alter_table=ON rebuilds the partitioned table and attempts to move data from the dropped partition to another partition with a compatible PARTITION ... VALUES definition. Data that cannot be moved to another partition is deleted.

MySQL Basic Table Partitioning Methods

Hash Table Partitioning

Partitioning by HASH is used mostly to achieve an even distribution of data between the partitions. You must specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided when creating the partitioned table.

You must use an SQL expression that returns an integer for the HASH expression. The only permitted data types beside integer are date types and one of the following functions:

```
ABS, CEILING, DAY, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, DATEDIFF, EXTRACT, FLOOR, HOUR,
MICROSECOND, MINUTE, MOD, MONTH, QUARTER, SECOND,
```

```
TIME_TO_SEC, TO_DAYS, TO_SECONDS, UNIX_TIMESTAMP (with TIMESTAMP columns), WEEKDAY,
YEAR, YEARWEEK
```

For other column types you can use KEY Partitioning, which takes any column used as part or all of the table's primary key.

Examples

Create a hash-partitioned Table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    INT NOT NULL,
   EVENT_DATE  DATE   NOT NULL,
   EVENT_STR   VARCHAR(500),
   ERROR_CODE  INT)
  PARTITION BY HASH (ERROR_CODE)
  PARTITIONS 3;
```

Create a key-partitioned Table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    INT NOT NULL,
   EVENT_DATE  DATE   NOT NULL,
   EVENT_STR   VARCHAR(500),
   ERROR_CODE  VARCHAR(10) PRIMARY KEY)
  PARTITION BY KEY ()
  PARTITIONS 3;
```

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/partitioning-hash.html>
- <https://dev.mysql.com/doc/refman/5.7/en/partitioning-key.html>

List Table Partitioning

As with the HASH partition, this partitioned column must be INT. To use LIST on varchar, use LIST COLUMNS.

Examples

Create a list-partition table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO    INT NOT NULL,
   EVENT_DATE  DATE   NOT NULL,
   EVENT_STR   VARCHAR(500),
   ERROR_CODE  INT)
  PARTITION BY LIST (ERROR_CODE)
  (PARTITION warning VALUES IN (3345, 5423,3332),
   PARTITION critical VALUES IN (9786, 9231, 6321));
```

Create a listed-columns-partition table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO      INT NOT NULL,
  EVENT_DATE    DATE   NOT NULL,
  EVENT_STR     VARCHAR(500),
  ERROR_CODE    VARCHAR(500))
  PARTITION BY LIST COLUMNS (ERROR_CODE)
  (PARTITION warning VALUES IN ('err1', 'err2', 'err3'),
  PARTITION critical VALUES IN ('err4', 'err5', 'err6'));
```

For more details, see: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-list.html>

Range Table Partitioning

Similar to a LIST partition, a RANGE partition can be use on integer values or RANGE COLUMNS for DATE or DATETIME.

Examples

Create a range-partitioned table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO      INT NOT NULL,
  EVENT_DATE    DATE   NOT NULL,
  EVENT_STR     VARCHAR(500))
  PARTITION BY RANGE (YEAR(EVENT_DATE))
  (PARTITION p0 VALUES LESS THAN (2015),
  PARTITION p1 VALUES LESS THAN (2016),
  PARTITION p2 VALUES LESS THAN (2017));
```

Create a ranged columns-partitioned table.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO      INT NOT NULL,
  EVENT_DATE    DATE   NOT NULL,
  EVENT_STR     VARCHAR(500))
  PARTITION BY RANGE COLUMNS (EVENT_DATE)
  (PARTITION p0 VALUES LESS THAN ('2015-01-01'),
  PARTITION p1 VALUES LESS THAN ('2016-01-01'),
  PARTITION p2 VALUES LESS THAN ('2017-01-01'));
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/partitioning-range.html>

Composite Table Partitioning – Sub-Partitioning Tables

With composite partitioning, a table can be partitioned by one data distribution method, and then each partition can be further subdivided into sub-partitions using the same, or different, data distribution method(s).

In MySQL 5.7, it is possible to subpartition tables that are partitioned by RANGE or LIST. Subpartitions may use either HASH or KEY partitioning.

You can either:

- Specify only the number of subpartitions for each partition
- Explicitly define subpartitions in any partition individually, this options is useful if you want to pick the names for your subpartitions

Note: All partitions must have the same number of subpartitions.

Examples

Create a subpartition RANGE-KEY with all partitions having two subpartitions.

```
CREATE TABLE EMPLOYESS
  (DEPARTMENT_ID INT NOT NULL,
   LAST_NAME     VARCHAR(50) NOT NULL,
   FIRST_NAME    VARCHAR(50),
   PRIMARY KEY (DEPARTMENT_ID, LAST_NAME))
  PARTITION BY RANGE (DEPARTMENT_ID)
  SUBPARTITION BY KEY (last_name)
  SUBPARTITIONS 2
  (PARTITION p1 VALUES LESS THAN (10),
   PARTITION p2 VALUES LESS THAN (20),
   PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/partitioning-subpartitions.html>.

Split and Exchange Partitions

Split Partitions

Oracle's SPLIT PARTITION STATEMENT translates to REORGANIZE PARTITION in MySQL. Create a LIST partition and then split one of the partitions.

RANGE partitions can be split at the last partition only.

```
CREATE TABLE SYSTEM_LOGS
  (EVENT_NO     INT NOT NULL,
   EVENT_DATE   DATE   NOT NULL,
   EVENT_STR    VARCHAR(500),
   ERROR_CODE   VARCHAR(500))
  PARTITION BY LIST COLUMNS (ERROR_CODE)
  (PARTITION warning VALUES IN ('err1', 'err2', 'err3'),
   PARTITION critical VALUES IN ('err4', 'err5', 'err6'));

ALTER TABLE SYSTEM_LOGS REORGANIZE PARTITION warning INTO
  (PARTITION warning0 VALUES IN ('err2.5', 'err3.5'),
   PARTITION warning1 VALUES IN ('err2.8', 'err3.8'));
```

For more information <https://dev.mysql.com/doc/refman/5.7/en/partitioning-management-range-list.html>

Exchange Partitions

Similar to Oracle, you can exchange tables with partitions.



```
ALTER TABLE orders
  EXCHANGE PARTITION p_ord3 WITH TABLE orders_year_2016;
```

For more details, see <https://dev.mysql.com/doc/refman/5.7/en/partitioning-management-exchange.html>.

Summary

Oracle Table Partition Type	Build-In MySQL Support	Link To Example
List	Yes	https://dev.mysql.com/doc/refman/5.7/en/partitioning-list.html
Range	Yes	https://dev.mysql.com/doc/refman/5.7/en/partitioning-range.html
Hash	Yes	https://dev.mysql.com/doc/refman/5.7/en/partitioning-hash.html
Composite Partitioning (sub Partitioning)	Yes	https://dev.mysql.com/doc/refman/5.7/en/partitioning-sub-partitions.html
Interval Partitioning	No	https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
Partition Advisor	No	https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
Preference Partitioning	No	https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
Virtual Column Based Partitioning	No	https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
MySQL Partitioning Automatic List Partitioning	No	https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
Split / Exchange Partitions	Yes	https://dev.mysql.com/doc/refman5.7/en/alter-table-partition-operations.html https://dev.mysql.com/doc/refman/5.7/en/partitioning-management-exchange.html

Sharding

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL doesn't support sharding

Oracle Usage



Sharding is a method of data architecture where table data is horizontally partitioned across independent databases. These databases are called shards. All of the shards make up a single logical database, which is referred to as a sharded database (SDB). Sharding a table is process of splitting this table between different shards where each shards will have sharded table with the same structure but different subset of rows.

Oracle 18c introduces following sharding enhancements:

- User-defined sharding. Before Oracle 18c data was redirected across shards by system. With user-defined sharding users are now able to explicitly redirect sharded table data to specific individual shards.
- Using JSON, BLOB, CLOB and spatial objects functionality in a sharded environment. These objects can now be used in sharded tables.

For additional details, see: <https://docs.oracle.com/en/database/oracle/oracle-database/18/shard/sharding-overview.html#GUID-0F39B1FB-DCF9-4C8A-A2EA-88705B90C5BF>

Migrate to Aurora MySQL

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
		N/A	MySQL doesn't support sharding


There is no equivalent option in MySQL, the most equivalent option will be to create application level sharding management that will interact with data that is spread across multiple instances.

Another option will be to assess the requirements and probably use another data store like Redshift / EMR / DynamoDB.

Monitoring



Oracle V\$ Views and the Data Dictionary vs. MySQL System Catalog & SHOW command

Feature Compatibility	SCT/DMS Automation Level	SCT Action Code Index	Key Differences
	N/A	N/A	Table names in queries need to be changed when using MySQL

Oracle Usage

Oracle provides several built-in views used to monitor the database and query its operational state. These views can be used to track the status of the database, view information about database schema objects, and obtain other information.

The data dictionary is a collection of internal tables and views that supply information about the state and operations of an Oracle database including database status, database schema objects (tables, views, sequences, etc.), users and security, and physical database structure (datafiles). The contents of the data dictionary are persisted to disk.

Examples of data dictionary views include:

- **DBA_TABLES:** Information about all tables in the current database.
- **DBA_USERS:** Information about all database users.
- **DBA_DATA_FILES:** Information about all physical data files in the database.
- **DBA_TABLESPACES:** Information about all tablespaces in the database.
- **DBA_TAB_COLS:** Information about columns (for all tables) in the database.

Note: Data dictionary view names can start with `DBA_*`, `ALL_*`, `USER_*`, depending on the level and scope of information presented (user-level versus database-level).

For the complete list of `dba_*` data dictionary views, see [Catalog Views / Data Dictionary Views](#)

Dynamic Performance Views (V\$ Views) are a collection of views that provide real-time monitoring information about the current state of the database instance configuration, runtime statistics, and operations. These views are continuously updated while the database is running.

Information provided by the dynamic performance views includes session information, memory usage, progress of jobs and tasks, SQL execution state and statistics, and various other metrics.

Common dynamic performance views include:

- **V\$SESSION:** Information about all current connected sessions in the instance.
- **V\$LOCKED_OBJECT:** Information about all objects in the instance on which active “locks” exist.
- **V\$INSTANCE:** Dynamic instance properties.
- **V\$SESSION_LONG_OPS:** Information about certain “long running” operations in the database such as queries currently executing.
- **V\$MEMORY_TARGET_ADVICE:** Advisory view on how to size the instance memory, based on instance activity and past workloads.

For additional details, see:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/data-dictionary-and-dynamic-performance-views.html#GUID-BDF5B748-EB43-4B48-938E-89099069C3BB>

MySQL Usage

MySQL provides two different ways to retrieve information about the state of the database and current activities. The information is similar in nature to the Oracle data dictionary tables and V\$ performance views. In addition, Amazon Aurora MySQL provides a “Performance Insights” console for monitoring and analyzing database workloads and troubleshooting performance issues.

Information Schema Tables

The information schema consists of views containing information about objects that were created in the current database.

- The information schema is specified by the SQL standard and is supported by MySQL.
- Some of these tables are comparable to Oracle USER_* Data Dictionary tables.
- The owner of this schema is the initial database user.
- Since the information schema is defined as part of the SQL standard, it can be expected to remain stable across MySQL versions.

SHOW Command

The SHOW command provides information about databases, tables, columns, and status information about the server.

- If the syntax for a SHOW statement includes a LIKE 'pattern' part, 'pattern' is a string that can contain the SQL % and _ wildcard characters. The pattern is useful for restricting statement output to matching values.
- The SHOW command has more dynamic views such as PROCESSLIST.
- Users must have PROCESS privilege to query this data.

Summary

Information	Oracle	MySQL
Database properties	V\$DATABASE	pg_database
Database sessions	V\$SESSION	SHOW PROCESSLIST
Database users	DBA_USERS	mysql.user
Database tables	DBA_TABLES	information_schema.TABLES
Database data files	DBA_DATA_FILES	information_schema.FILES
Table columns	DBA_TAB_COLS	information_schema.COLUMNS
Database locks	V\$LOCKED_OBJECT	information_schema.INNODB_LOCKS

Information	Oracle	MySQL
Currently configured runtime parameters	V\$PARAMETER	SHOW GLOBAL VARIABLES
All system statistics	V\$SYSSTAT	information_schema.INNODB_METRICS
Privileges on tables	DBA_TAB_PRIVS	information_schema.TABLE_PRIVILEGES
Information about IO operations	V\$SEGSTAT	SHOW STATUS LIKE '%read%'; SHOW STATUS LIKE '%write%';

For more details, see:

- <https://dev.mysql.com/doc/refman/5.7/en/show.html>
- <https://dev.mysql.com/doc/refman/5.7/en/information-schema.html>

Migration Quick Tips



Migration Quick Tips

This section provides migration tips that can help save time as you transition from Oracle to Aurora MySQL. It addresses many of the challenges faced by administrators new to Aurora MySQL. Some of these tips describe functional differences in similar features between Oracle and Aurora MySQL.

Management

- In Aurora MySQL, the term "Database Snapshot" is equivalent to Oracle's RMAN backup.
- Partitioning in Aurora MySQL does not provide many of Oracle's management features such as Partition Advisor, Preference Partitioning, Virtual Column-Based Partitioning, and Automatic List Partitioning.
- Unlike Oracle's statistics, Aurora MySQL does not collect detailed key/value distributions in tables; it only collects statistics on indexes.
- Amazon services, such as Lambda, can be used to replicate functionality of features not provided by MySQL (for example, email).
- Amazon RDS manages parameters and backups. It is very useful for checking a parameter's value against its default and comparing them to another parameter group.
- With just a few clicks, you can create Replicas to implement High Availability.
- Aurora MySQL does not have an equivalent to database links. It can only query across databases within the same instance.

SQL

- Aurora MySQL does not support statement-level triggers or triggers on system events.
- Aurora MySQL does not support many cursor status checks. When declaring cursors in Aurora MySQL, you must create an explicit HANDLER object.
- To execute a stored procedure (function), use CALL instead of EXECUTE.
- To execute a string as a query, use Aurora MySQL Prepared Statements instead of EXECUTE(<String>) syntax.
- In Aurora MySQL, IF blocks must be terminated with END IF. WHILE..LOOP loops must be terminated with END LOOP.
- Unlike Oracle, Aurora MySQL auto-commit defaults to ON, be sure to set it to OFF if you want database behavior similar to Oracle.
- Similar to Oracle, collations can be defined at the server, database, and column level. They cannot be defined at the table level.
- Oracle's DELETE <Table Name> syntax, which allows omitting the FROM keyword, is invalid in Aurora MySQL. Add the FROM keyword to all DELETE statements.
- Aurora MySQL's AUTO_INCREMENT column property is similar to IDENTITY in Oracle.
- Error handling in Aurora MySQL has less features than Oracle. For special requirements, you can log or send alerts by inserting into tables or catching errors.
- Aurora MySQL does not support the MERGE statement. Use the REPLACE statement and the INSERT... ON DUPLICATE KEY UPDATE statement as alternatives.
- Unlike Oracle, You cannot concatenate strings in Aurora MySQL using the "||" operator.

- Aurora MySQL is much stricter than Oracle for statement terminators. Be sure to always use a semicolons at the end of statements.
- Aurora MySQL does not support BFILE, ROWID, UROWID data types.
- In MySQL, temporary tables are retained only for the session and only the session that created a temporary table can query it.
- MySQL does not support unused or virtual columns and there is no workaround for replacing unused columns to achieve functionality similar to virtual columns. Views and functions can be combined.
- MySQL does not support Materialized Views, use Views or summary tables instead.
- Explore AWS to locate features that can be replaced with Amazon services. They can help you maintain your database and decrease costs.
- MySQL architecture allows multiple databases in a single instance and can be useful for consolidation projects.
- Beware of control characters when copying and pasting a script to Aurora MySQL clients. Aurora MySQL is much more sensitive to control characters than Oracle and can result in frustrating syntax errors that are hard to find.

Glossary

ACID

Atomicity, Consistency, Isolation, Durability

AES

Advanced Encryption Standard

ANSI

American National Standards Institute

API

Application Programming Interface

ARN

Amazon Resource Name

AWS

Amazon Web Services

BLOB

Binary Large Object

CDATA

Character Data

CLI

Command Line Interface

CLOB

Character Large Object

CLR

Common Language Runtime

CPU

Central Processing Unit

CRI

Cascading Referential Integrity

CSV

Comma Separated Values

CTE

Common Table Expression

DB

Database

DBCC

Database Console Commands

DDL

Data Definition Language

DEK

Database Encryption Key

DES

Data Encryption Standard

DML

Data Manipulation Language

DQL

Data Query Language

FCI

Failover Cluster Instances

HADR

High Availability and Disaster Recovery

IAM

Identity and Access Management

IP

Internet Protocol

ISO

International Organization for Standardization

JSON

JavaScript Object Notation

KMS

Key Management Service

NUMA

Non-Uniform Memory Access

OLE

Object Linking and Embedding

OLTP

Online Transaction Processing

PaaS

Platform as a Service

PDF

Portable Document Format

QA

Quality Assurance

RDMS

Relational Database Management System

RDS

Amazon Relational Database Service

REGEXP

Regular Expression

SCT

Schema Conversion Tool

SHA

Secure Hash Algorithm

SLA

Service Level Agreement

SMB

Server Message Block

SQL

Structured Query Language

SQL/PSM

SQL/Persistent Stored Modules

SSD

Solid State Disk

SSH

Secure Shell

T-SQL

Transact-SQL

TDE

Transparent Data Encryption

UDF

User Defined Function

UDT

User Defined Type

UTC

Universal Time Coordinated

XML

Extensible Markup Language