ORACLE University

Integrated Cloud Applications & Platform Services

MySQL for Database Administrators

Student Guide - Volume II

D61762GC50

Edition 5.0 | June 2019 | D106725

Learn more from Oracle University at **education.oracle.com**

ORACLE

Authors

KimSeong Loh
Mark Lewin

Technical Contributors and Reviewers

Frederic Descamps
Hananto Wicaksono
Igor Ilyin
Mirko Ortensi

Editors

Adrita Biswas
Aju Kumar
Raj Kumar
Moushmi Mukherjee

Graphic Editors

Kavya Bellur
Pushparaj Kundar

Publishers

Pavithran Adka
Veena Narasimhan

Contents

4 Configuring MySQL

9

Optimizing Query Performance

Objectives

After completing this lesson, you should be able to:

- Identify queries suitable for optimization
- Examine query index usage
- Create indexes to improve server performance
- Maintain index statistics

Topics

- Identifying slow queries
- `EXPLAIN` statement
- Working with indexes
- Index statistics

Identifying Slow Queries

- Identify the queries that the server executes:
  - Use the general query log and slow query log.
- Identify queries that take a long time:
  - Use the slow query log.
  - Prioritize these over outlier queries or queries that you execute infrequently.
- Find queries that execute many times:
  - Regularly execute **`SHOW PROCESSLIST`** to see currently executing statements and their durations and to identify emerging patterns.
  - Use **`sys.statement_analysis`** to view normalized statements with aggregated statistics.
  - Use the slow query log with a low threshold to record most statements.

Choosing What to Optimize

- Do not assume that you need to optimize only the slowest queries.
  - You might improve the overall performance of the system by optimizing the most frequently executed queries, even if such queries are already executing relatively quickly.
    - Optimizing a frequent query (even by only a small amount) can free up resources, reduce locking, and improve the response time and throughput of the server.
    - Optimizing an infrequent query provides less overall benefit.
- Consider the following examples:
  - You optimize a query that takes two seconds so that it now takes 800 ms, but this query executes on average once per minute.
  - You optimize a query that takes 20 ms so that it now takes 15 ms, but this query executes several thousand times per minute.

If the server spends an extra 5 ms on a query that executes several thousand times per minute, that might translate to 15 or 20 seconds of cumulatively delayed statement execution on that server. Small optimizations on frequent queries might give more overall improvement.

Topics

- Identifying slow queries
- `EXPLAIN` statement
- Working with indexes
- Index statistics

Using `EXPLAIN` to See Optimizer's Choice of Index

- Produces a query execution plan showing how the optimizer plans to execute a given SQL statement
  - Includes information from MySQL server optimizations
- Examines `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, and `DELETE` statements
- Does not return any data from the data sets; does not perform any data modification in the statement
- Chooses optimal operations based on:
  - Query
  - Structure of tables in the query
  - Any relevant indexes

EXPLAIN: Example

```
mysql> EXPLAIN SELECT * FROM employees WHERE emp_no=10001\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (#.## sec)
```


The output in the slide generates a warning. When you use `EXPLAIN` to view the plan for a `SELECT` statement, it produces a `note` level event that describes query rewriting and optimization actions.

When the `sql_notes` system variable is set to `1` (the default), `note`-level events are treated as warnings. For more information, see the section titled “Displaying Query Rewriting and Optimization Actions” later in this lesson.

EXPLAIN Output

- **id**: Number identifying the examined statement
- **select_type**: Type of select used in the query
  - SIMPLE: The query does not use UNION or subqueries.
  - Other values indicate the type of union or subquery.
  - For DML statement: INSERT, REPLACE, UPDATE, and DELETE
- **table**: Table for the output row
- **partitions**: Partitions that the optimizer needs to examine to execute the query
- **type**: Index or join comparison type
- **possible_keys**: Indexes that are relevant to the query
- **key**: Specific index chosen by the optimizer

EXPLAIN Output

- **key_len**: Size (in bytes) of the left-most columns used to search the index
- **ref:** Columns (or const) compared to the index
- **rows**: Estimated number of rows that the optimizer predicts will be returned by the query
- **filtered:** Percentage of rows that are filtered by the table condition
- **Extra**: Additional per-query information provided by the optimizer or storage engine

Common `type` Values

The `type` column indicates the type of comparison used by the optimizer to access rows.

- **`ALL`**: Full table scan
- **`index`**: Full index scan
- **`const`**: Matching a primary or unique key against a constant at the start of a query
- **`eq_ref`**: Matching a single referenced value (identified by the ref column) for equality
- **`ref`**: Matching one or more referenced values for equality
- **`range`**: Matching rows in a range that the named index (key) supports

Displaying Query Rewriting and Optimization Actions

- Use `SHOW WARNINGS` to display further details about optimizations.
- Each displayed message provides extended information about the optimizer's plan for the query.
- It shows a rephrased version of the query in pseudo-SQL that represents the optimized sequence of operations, for example:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select '10001' AS `emp_no`,'1953-09-02' AS
`birth_date`,'Georgi' AS `first_name`,'Facello' AS `last_name`,'M' AS
`gender`,'1986-06-26' AS `hire_date` from `employees`.`employees` where 1
1 row in set (#.## sec)
```

- `SHOW WARNINGS` also shows any messages that are specific to the storage engine's own optimizations, if any.

`EXPLAIN` produces `note`-level messages that have the error code 1003. That error code has the internal symbol `ER_YES` and the associated error name `YES`. When `sql_notes` is set to `1`, `note`-level messages are treated as warnings. Although they are not warnings or errors in the usual sense, they enable you to view these extra messages using the client as an interface.

Some other storage engines, such as NDB, add messages to the output of `SHOW WARNINGS` that indicate storage engine–specific optimizations.

EXPLAIN Example: Table Scan

```
mysql> EXPLAIN SELECT *
    -> FROM employees
    -> WHERE first_name='Mong'\G
*********** 1. row **************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 299246
     filtered: 10.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

- The `possible_keys`, `key`, and `key_len` columns display **`NULL`**.
  - The query cannot use any indexes to improve the query's performance.
- The `type` column shows the value **`ALL`**, indicating a table scan.
- The `rows` column shows the number `299246`.
  - This is InnoDB's *estimate* of the number of rows in the table.

ORACLE

InnoDB generates an estimate of the number of rows in a table or an index when you run `ANALYZE TABLE` *`tablename`*.

EXPLAIN Example: Primary Key

```
mysql> EXPLAIN SELECT *
    -> FROM employees
    -> WHERE emp_no=10001\G
*********** 1. row **************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (#.## sec)
```

- The type of the operation is **const**.
  - const queries match literal (constant) values against primary or unique keys.
- The chosen index is shown as the key value PRIMARY, with a size of four bytes.
  - This index is compared for equality with a referred constant value, shown by ref: const.
  - Comparing the primary key with the specified value produces at most a single row.
- In addition, the optimizer estimates that it needs to examine only one row, shown by the value in the rows column.

EXPLAIN Example: Non-unique Index

```
mysql> EXPLAIN SELECT emp_no
    -> FROM dept_manager
    -> WHERE dept_no='d001'\G
*********** 1. row ***********
           id: 1
  select_type: SIMPLE
        table: dept_manager
   partitions: NULL
         type: ref
possible_keys: PRIMARY,emp_no,dept_no
          key: dept_no
      key_len: 16
          ref: const
         rows: 2
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (#.## sec)
```

- The `type` of the operation is **`ref`**.
  - Compares the column value with a referred value (a `const`)
  - Matches the referenced value with a nonunique column
    - Might match multiple rows
- The optimizer estimates the number of rows to be two.
  - The actual value depends on the data.
- The `Extra` text `Using index` indicates that the query does not result in an additional seek to read the actual row's data.

The structure of the `dept_manager` table is as follows:

```
CREATE TABLE dept_manager (
   dept_no      CHAR(4)          NOT NULL,
   emp_no       INT              NOT NULL,
   from_date    DATE             NOT NULL,
   to_date      DATE             NOT NULL,
   KEY          (emp_no),
   KEY          (dept_no),
   FOREIGN KEY (emp_no)  REFERENCES employees (emp_no)    ON
DELETE CASCADE,
   FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON
DELETE CASCADE,
   PRIMARY KEY (emp_no,dept_no)
);
```

All the secondary index in InnoDB tables also contain the primary key values.

The `dept_no` index also contain the values of `emp_no`. Therefore, the output shows a "`Using index`" in the `Extra` column.

EXPLAIN and Complex Queries

- EXPLAIN produces multiple output rows when you analyze a complex query.
  - A complex query is a query that contains subqueries, UNION clause, or JOIN clause.
- Each row uses an **id** to uniquely identify the statement that it refers to.
  - If you have a SELECT statement with a subquery or UNION that has its own SELECT statement, you will have two rows, each with a different id.
    - Each row refers to table operations in a separate statement.
  - For a single SELECT statement that joins two tables, you get two rows with the same id.
    - Both rows refer to table operations within the same statement.

EXPLAIN Example: Simple Join

```
mysql> EXPLAIN SELECT first_name,
    -> last_name, title
    -> FROM employees
    -> JOIN titles USING(emp_no)
    -> WHERE title='Senior Engineer'\G
************* 1. row *************
           id: 1
  select_type: SIMPLE
        table: titles
   partitions: NULL
         type: index
possible_keys: PRIMARY,emp_no
          key: emp_no
      key_len: 4
          ref: NULL
         rows: 417756
     filtered: 10.00
        Extra: Using where; Using index
************** 2. row **************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.titles.emp_no
         rows: 1
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (#.## sec)
```



```
CREATE TABLE employees (
    emp_no      INT             NOT NULL,
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,
    hire_date   DATE            NOT NULL,
    PRIMARY KEY (emp_no)
);

CREATE TABLE titles (
    emp_no      INT             NOT NULL,
    title       VARCHAR(50)     NOT NULL,
    from_date   DATE            NOT NULL,
    to_date     DATE,
    KEY         (emp_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,title, from_date)
);
```

Explanation of Simple Join Output

When you join tables in a query, the optimizer must perform an operation on each table.

- The join in the preceding slide has a `WHERE` clause that references an unindexed field.
- The first operation is an index scan of all rows in the `emp_no` index of the titles table.
  - It reads all the index entries, and filters them based on the `WHERE` clause against the primary key value. All secondary indexes in InnoDB tables also contain the primary key values.
- The second operation references the first one:
  - The type of the second operation is `eq_ref`, indicating that it matches each row for equality with its compared value.
  - The value in question is the qualified column name `employees.titles.emp_no`, indicating that the value to compare is not a constant provided in the statement, but comes from that other column.

Index Types

- **Nonunique**: Values can appear multiple times in the index.
  - This is the default index type.
- **Unique**: Values must be unique (or `NULL`).
- **Primary key**: Values are unique and cannot contain `NULL`.
  - Every table has at most one primary key.
- **Fulltext**: Values are character string data, and the index supports full text searching.
- **Spatial**: Values are spatial data types such as `GEOMETRY`, `POINT`, or `POLYGON`.

The index types shown in the slide are supported by InnoDB. Other storage engines might not support these index types.

Functions used in full text searching and spatial data handling are covered in the course titled *MySQL for Developers.*

MySQL Enterprise Monitor Query Analyzer

- Monitors SQL statements executed on the MySQL server:
  - Nature of query
  - Number of executions
  - Execution times
- Extracts this information from the Performance Schema by using the MySQL Enterprise Monitor Agent by default.
  - Can configure client applications to route database requests via MySQL Proxy and MySQL Aggregator
  - Can install a Connector plugin to send this information directly to the MySQL Enterprise Monitor Service Manager
- Normalizes queries for easier reporting
  - Combines similar queries with different literal values
- Enables you to drill into each query for detailed information

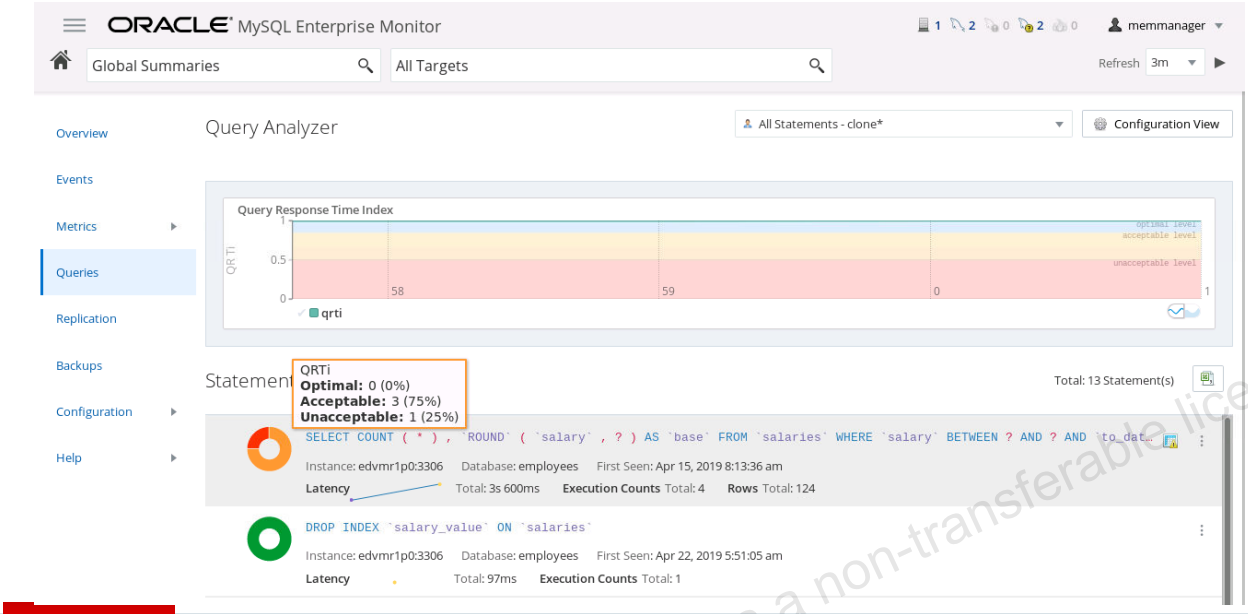
Query Response Time Index

MySQL Enterprise Monitor Query Analyzer generates a Query Response Time Index (QRTi) for each query.

- Enables an “at-a-glance” indicator of query performance
- Is generated by an algorithm that considers number of executions and duration of each query:

Status	Default time value	Assigned Value	Meaning
Optimum	100ms	1.0 (100%)	All instances of the query were within the acceptable response time.
Acceptable	4 * Optimum: 100ms to 400ms	0.5 (50%)	Query execution time was less than 400 ms (the default acceptable response time).
Unacceptable	> Acceptable	0 (0%)	All instances of the query exceeded the acceptable response time.

Query Analyzer User Interface



Topics

- Identifying slow queries
- `EXPLAIN` statement
- Working with indexes
- Index statistics

Creating Indexes to Improve Query Performance

- Enables efficient access to data rows
  - A query that searches for a value in an indexed field can immediately find rows containing that value.
    - A seek, which is efficient
  - A query that searches for a value in an unindexed field must read all rows to find rows containing that value.
    - A scan, which is inefficient
- Supports the following operations:
  - Direct value matching
    - Example: Find words such as “xenolith.”
  - Existence checking
    - Example: Determine that the word “xzzq” does not exist.
  - Range scans
    - Example: Find all words that begin with “xy.”

Consider the following analogy: If you are staying in a large hotel and you ask the receptionist if you have any printed messages, he or she looks in the specific mailbox assigned to your room to find them.

- If there is a message for you, the receptionist finds it in that specific location (a *seek*).
- If there is no message for you, that mailbox is empty. The receptionist does not need to scan all mailboxes for messages that are addressed to you.

Similarly, if you want to find all words in a dictionary that contain the letter *x*, you would need to go through all of the definitions (a *scan*). Alternatively, if you want to find all words beginning with the letter *x*, that is a much more efficient operation because all such words appear together in the dictionary. The dictionary is indexed in word order, with each word acting as the index key.

Creating and Dropping Indexes on Existing Tables

- To change a table's primary key:

```
ALTER TABLE table DROP PRIMARY KEY,
ADD PRIMARY KEY(col1, col2);
```

- To add a unique key:

```
ALTER TABLE table ADD UNIQUE (col3);
CREATE UNIQUE INDEX index2 ON table(col4);
```

- To add an ordered (nonunique) index with a name:

```
ALTER TABLE table ADD INDEX (col5);
CREATE INDEX index3 ON table(col6);
```

- To drop a nonprimary key by name:

```
ALTER TABLE table DROP INDEX indexname;
DROP INDEX indexname ON table;
```


`CREATE INDEX` and `DROP INDEX` are internally mapped to `ALTER TABLE` statements. You cannot use `CREATE INDEX` to create a primary key.

Displaying Indexes with SHOW INDEXES FROM

```
mysql> SHOW INDEXES FROM
                  departments\G
************ 1. row **************
        Table: departments
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: dept_no
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
************ 2. row **************
        Table: departments
   Non_unique: 0
     Key_name: dept_name
 Seq_in_index: 1
  Column_name: dept_name
    Collation: A
  Cardinality: 9
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
2 rows in set (#.## sec)
```

The `INFORMATION_SCHEMA.STATISTICS` view contains similar information to the information returned by the `SHOW INDEXES` statement.

Example: To view information about indexes in the `employees.departments` table like that shown in the slide, execute the following statement:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA='employees' AND TABLE_NAME='departments'\G
```

Topics

- Identifying slow queries
- `EXPLAIN` statement
- Working with indexes
- Index statistics

Maintaining InnoDB Index Statistics

- MySQL's optimizer uses index key distribution statistics to decide:
  - Which indexes MySQL uses for a specific table within a query
  - The join order when you perform a join on something other than a constant
- Update index statistics:
  - Automatically after 10% of rows change
  - Manually with **`ANALYZE TABLE`**

Automatically Updating Index Statistics

- InnoDB stores statistics persistently.
  - Enabled by default with the **`innodb_stats_persistent`** variable
  - Enabled per table with the **`STATS_PERSISTENT`** table option
  - Stored in the **`mysql.innodb_index_stats`** table
- Statistics update automatically.
  - Sampling 20 data pages by default
    - Changed with the **`innodb_stats_persistent_sample_pages`** variable
  - After 10% of rows change, enabled by default with the **`innodb_stats_auto_recalc`** variable
  - Optionally, after executing metadata statements such as `SHOW TABLE STATUS`, or when querying `INFORMATION_SCHEMA.TABLES`, enabled with the **`innodb_stats_on_metadata`** variable

If you disable persistent statistics gathering for one or more tables, InnoDB stores statistics for those tables in a volatile form internally, and uses the value of the `innodb_stats_transient_sample_pages` variable to decide how many pages it reads.

Using ANALYZE TABLE

- Analyzes and stores the key distribution statistics of a table
- Is used to make better choices about query execution
- Works with InnoDB, NDB, and MyISAM tables
- Supports partitioned tables
- ANALYZE TABLE option:
  - NO_WRITE_TO_BINLOG or LOCAL: Suppress binary log
- Example of an ANALYZE TABLE result:

```
mysql> ANALYZE LOCAL TABLE salaries;
+--------------------+---------+----------+----------+
| Table              | Op      | Msg_type | Msg_text |
+--------------------+---------+----------+----------+
| employees.salaries | analyze | status   | OK       |
+--------------------+---------+----------+----------+
1 row in set (#.## sec)
```

ORACLE

ANALYZE TABLE locks the table with a read lock for the duration of the analysis.

Rebuilding Indexes

- Use the **FORCE**:

```
ALTER TABLE tablename FORCE
```

  - Forces a rebuild even though the statement does not change the table structure
  - Reorganizes data more evenly across data pages
  - Reclaims file system space used by empty pages
- Rebuild full text indexes by using **OPTIMIZE TABLE**:

```
SET innodb_optimize_fulltext_only = 1;
OPTIMIZE TABLE tablename;
```

  - By default, **OPTIMIZE TABLE** rebuilds the entire table.
  - The **innodb_optimize_fulltext_only** option ensures that `OPTIMIZE TABLE` rebuilds only the full text index.

Reorganize `MyISAM` and `ARCHIVE` tables by using the `OPTIMIZE TABLE` statement. On `InnoDB`, this statement executes by mapping to the `ALTER TABLE` statement shown in the slide.

mysqlcheck Client Program

- Can be more convenient than issuing SQL statements
- Works with InnoDB, MyISAM, and ARCHIVE tables
- Three levels of checking:
  - Table specific
  - Database specific
  - All databases
- Some mysqlcheck maintenance options:
  - **--analyze**: Performs ANALYZE TABLE
  - **--check**: Performs CHECK TABLE (default)
  - **--optimize**: Performs OPTIMIZE TABLE
- Examples:

```
mysqlcheck -uroot -p employees employees salaries
mysqlcheck --login-path=admin --analyze --all-databases
```

Invisible Indexes

- Enable you to “hide” indexes from the optimizer
- Test the effect of removing indexes on query performance
  - Without making destructive changes to drop the index.
  - Avoid expensive operation to re-create the index if it is found to be required.
- Continue to be updated by the server when data is modified by DML statements
- Cannot be applied to primary keys
- Should be marked as `INVISIBLE` in `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX` statements
  - Can be “unhidden” by using the `VISIBLE` keyword

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. By marking an index as invisible, you are effectively "hiding" it from the optimizer while the index itself remains intact and can be restored at any time. This feature makes it much easier to test the removal of indexes and to perform a staged rollout of the changes.

Note that marking an index as invisible is not the same as disabling it. The index is still kept up-to-date and maintainable by DML statements.

Dropping and readding an index can be expensive for a large table, whereas making it invisible and visible are fast, in-place operations. Every index is visible by default. To mark it as invisible, you use the `INVISIBLE` keyword in a `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX` statement. To restore the index visibility, execute another such statement with the `VISIBLE` keyword instead.

Histograms

- Provide the optimizer with extra information about data distribution within nonindexed columns
  - The optimizer knows how data is distributed for index columns, but not columns that do not form part of an index.
  - Skewed data might affect query performance.
- Approximate the data distribution within those columns
- Enable the optimizer to make better decisions

To produce an optimal execution plan for a query, the optimizer benefits from knowing how the data is distributed in each column. It has some idea of this for columns that form part of an index, but it is clueless about the distribution of data in nonindexed columns. If a column contains skewed data, this can affect query execution.

You can create histograms on these columns to provide the optimizer an approximation of the data distribution within the columns. These help the optimizer to make more informed decisions about how to access the data they contain.

Example: The Query

```
mysql> EXPLAIN SELECT * FROM orders
       JOIN customer ON o_custkey = c_custkey
       WHERE o_orderdate < '1993-01-01' AND c_mktsegment = "AUTOMOBILE"\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
   partitions: NULL
         type: ALL
possible_keys: PRIMARY
          key: NULLz
      key_len: NULL
          ref: NULL
         rows: 149050
     filtered: 10.00
        Extra: Using where
*************************** 2. row ***************************
...
2 rows in set, 1 warning (#.## sec)
```

```
mysql> SELECT count(*) FROM orders
       JOIN customer ON o_custkey = c_custkey
       WHERE o_orderdate < '1993-01-01' AND c_mktsegment = "AUTOMOBILE"\G
*************************** 1. row ***************************
count(*): 45127
1 row in set (49.98 sec)
```

ORACLE

To understand how histograms affect query execution, consider this example. This is a query that joins the orders and customer tables for orders of automobiles in 1993 and later. Note from the `EXPLAIN` output in column one that you must perform a full table scan on the `CUSTOMER` table and the selectivity (`filtered` column in the `EXPLAIN` output) is 10%. The selectivity of a column is the "uniqueness" of its values. High selectivity implies high uniqueness, or a low number of matching values. Low selectivity implies a low uniqueness, or a high percent of matches. In this example, many rows in the `CUSTOMER` table have "AUTOMOBILE" in the `c_mktsegment` column. When you execute a count of the number of rows involved in the query, it takes approximately 50 seconds.

Example: Creating a Histogram

```
mysql> ANALYZE TABLE customer
    -> UPDATE HISTOGRAM ON c_mktsegment WITH 1024 BUCKETS;
+--------------+-----------+----------+------------------------------+
| Table        | Op        | Msg_type | Msg_text                     |
+--------------+-----------+----------+------------------------------+
| dbt3.customer | histogram | status   | Histogram statistics created |
|              |           |          | for column 'c_mktsegment'.   |
+--------------+-----------+----------+------------------------------+
1 row in set (#.## sec)
```


To provide the optimizer with more insight into the data distribution of the `c_mktsegment` column in the `CUSTOMER` table, create a histogram, You use the `ANALYZE TABLE ... UPDATE HISTOGRAM` syntax to do this, specifying a number of “buckets” for the data to be sorted into. A histogram sorts values into “buckets,” as you might sort coins into buckets. You can have up to 1,024 such buckets but creating too many is often counter-productive, so typically you would start with a low number first, then test and refine.

Example: The Query with Histogram Data

```
mysql> EXPLAIN SELECT * FROM orders
       JOIN customer ON o_custkey = c_custkey
       WHERE o_orderdate < '1993-01-01' AND c_mktsegment = "AUTOMOBILE"\G
*************************** 1. row ***************************
...
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: dbt3.orders.o_custkey
         rows: 1
     filtered: 19.84
        Extra: Using where
2 rows in set, 1 warning (#.## sec)
```

```
mysql> SELECT count(*) FROM orders
       JOIN customer ON o_custkey = c_custkey
       WHERE o_orderdate < '1993-01-01' AND c_mktsegment = "AUTOMOBILE"\G
*************************** 1. row ***************************
count(*): 45127
1 row in set (6.35 sec)
```

After creating the histogram, you can see that the optimizer chooses not to start with the `CUSTOMER` table because almost twice as many rows (19.84%) will cause look-ups into the `ORDERS` table.

This query now executes much faster: under 6.5 seconds compared to 50 seconds before creating the histogram.

Quiz

Which queries are most important to optimize?

a. Complex queries
b. Queries that cannot use a `WHERE` clause
c. Queries that execute most frequently
d. Queries that take the most time to execute

Answer: c

Summary

In this lesson, you should have learned how to:

- Identify queries suitable for optimization
- Examine query index usage
- Create indexes to improve server performance
- Maintain index statistics

Practices

- 9-1: Improving Query Performance with Indexes
- 9-2: Using MySQL Enterprise Monitor Query Analyzer

10

Choosing a Backup Strategy

Objectives

After completing this lesson, you should be able to:

- Distinguish between the different types of backup
- State the advantages and disadvantages of the various backup techniques
- Implement a backup strategy

Topics

- Understanding backups
- Backup techniques
- Creating a backup strategy

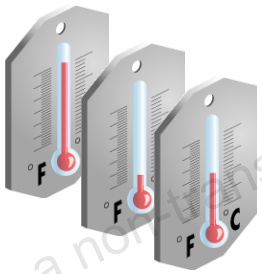
Reasons to Back Up

- Database recovery
  - Restoring from complete system failure
  - Recovering certain portions of the system to a state prior to a user error
- Auditing and analysis
  - Creating an environment separate from the primary production environment for data audit or analysis
- Typical DBA tasks
  - Transferring data from one system to another
  - Creating a development server based on a specific state of a production server

Backup Types

Backup types affect how applications interact with data during the backup operation:

- **Hot** backups generally allows applications full access to the data.
- **Cold** backups generally does not allow applications to access data.
- **Warm** backups permit applications to read, but not to modify data.



Hot Backups

- Hot backups take place while the data is being read and modified with little to no interruption of your ability to interface with or manipulate data.
- The system remains accessible for operations that read and modify data.
- Lock data by:
  - Using MVCC
  - Locking at a low level (such as row level)
  - Not locking at all so that applications can continue accessing the data

Multiversion concurrency control (MVCC) is a feature of InnoDB that uses the undo log to maintain a consistent snapshot of what the data looked like at a point in time.

Cold Backups

- Take place while the data is inaccessible to users
  - Typically the server is in an inaccessible mode or shut down entirely during a cold backup.
  - You cannot read or make any modifications to the data while the backup takes place.
- Prevent you from performing any activities with the data
- Do not interfere with the performance of the system when it is up and running
  - However, having to lock out or completely block user access to data for an extended period of time is not acceptable for some applications.

Warm Backups

- Take place while the data is being accessed
  - In most cases, the data cannot be modified while the backup is taking place.
- Have the advantage of not having to completely lock out end users
  - However, the disadvantage of not being able to modify the data sets while the backup is taking place can make this type of backup not suitable for certain applications.
- Might result in performance issues because you cannot modify data during the backup.
  - Updating users may be blocked for long duration of time.

Quiz

Which backup type takes place while the data is being read and modified with little to no interruption to your ability to interface or manipulate data?

a. Cold backup
b. Hot backup
c. Warm backup

Answer: b

Topics

- Understanding backups
- Backup techniques
- Creating a backup strategy

Backup Techniques

- Logical backup:
  - Textual representation: SQL statements
  - Results in a text file containing SQL statements to reconstruct the database
- Physical backup:
  - Binary copy of the MySQL database files
- Snapshot based
- Replication based
- Incremental backup:
  - Created by copying and flushing the MySQL binary logs

Logical Backups

Perform a complete data dump by using **`mysqldump`** or **`mysqlpump`**.

- These data dumps are based on a specific point in time but are the slowest of all the backup techniques.
- Advantage:
  - The process creates a SQL script that you can execute on a MySQL server.
  - You can use the script to reload the database on another host, running a different architecture.

- Disadvantage:
  - By default (and always for non-InnoDB tables), `mysqldump` and `mysqlpump` lock tables during the dump, which prevents users from modifying data during the backup.

Logical Backups

- Logical backups:
  - Convert databases and tables to SQL statements
  - Are portable
  - Require that the MySQL server is running during the backup
  - Can back up both local and remote MySQL servers
  - Are generally slower than raw (binary) backups
- The size of a logical backup file might exceed the size of the database being backed up.
  - The statements might take up more space than the data they represent.
  - However, because InnoDB stores data in data pages, which can contain free space, the InnoDB data files often take considerably more space on disk than the data requires.

Logical Backup Conditions

- Logical backups are warm backups:
  - The MySQL server must be running when you create logical backups.
  - Other applications can perform read operations during the logical backup.
  - The server creates the files by reading the structure and contents of the tables being backed up.
  - It then converts the structure and data to SQL statements.
- Using logical backups, you can back up local and remote MySQL servers.
  - Raw backups can be performed only on the server host.

Logical Backup Performance

- Logical backups are generally slower than raw backups.
  - The MySQL server must read tables and interpret their contents.
  - It must then translate the table contents to a disk file, or send the statements to a client program, which writes them out.
  - The `mysqlpump` client utility performs better than `mysqldump` because it processes databases and their objects in parallel, but it is still slower than a raw backup.
- Recovering from a logical backup is slower than recovering from a raw backup.
  - The recovery process executes the script containing the individual `CREATE` and `INSERT` statements that create each backed-up table and row.
  - All the indexes in the recovered table must be rebuilt.

Physical Backups

- Are created by copying the data files
  - Use standard commands such as `tar`, `cp`, `cpio`, `rsync`, or `xcopy`.
  - Use physical mirroring or block device online disk copying.
  - Use snapshot-based file archives.
- Must be restored to the same storage engine and MySQL version
  - When you recover a raw MySQL backup from an InnoDB table, it remains an InnoDB table on the target server.
- Can be restored across machine architectures
  - The files must be binary portable on the storage engine level.
- Are faster to perform and restore than logical backups and recoveries

Physical Backup Files

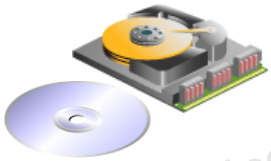
- Raw binary backups are faster than logical backups because the process is a simple file or file system copy.
- A physical backup is an exact representation of the bits in the database files.
  - These copies preserve the databases in exactly the same format in which MySQL itself stores them on disk.
  - Because they are exact copies of the originals, raw backups are the same size as the original.

Physical Backup Conditions

- The server must not modify data files during backup.
  - If you copy the live data files, prevent writes to those files:
    - For InnoDB: MySQL server shutdown is required.
    - For MyISAM: Lock tables to allow reads but not changes.
- You can also minimize the effect to MySQL and applications by using techniques that separate the data files being backed up from the MySQL server:
  - Snapshots
  - Replication
  - DRBD or other methods that copy the whole file system

Online Disk Copies

- You can back up data directly to another disk by using:
  - Processes such as replication or RAID mirroring
  - External applications such as DRBD
    - Distributed Replicated Block Device
    - Enables low-level disk copying between clustered computers
- These techniques provide
  - Live (or nearly live) backups
  - A quick means of recovering data in the event of a hardware failure



- Disadvantage:
  - Replicas are created in real time (or near real time), so you cannot use this technique to recover from data loss caused by user or application errors.

Snapshot-Based Backups

- Create a point-in-time copy of the data
  - You typically perform a raw backup against the snapshot copy.
- Provide a logically frozen version of the file system from which you can take MySQL backups
  - The frozen file system does not necessarily contain a consistent database image.
  - When you recover from such a file system, InnoDB must perform its own recovery to ensure there are no incomplete transactions.
- Greatly reduce the time during which the database and applications are unavailable

Performing a Snapshot

- Snapshot-based backups use a snapshot capability that is external to MySQL.
  - Example: If your data files are on an LVM2 logical volume on Linux that contains an appropriate file system such as ext4, you can create snapshot backups.
- Snapshot-based backups work best for transactional engines that perform their own recovery, such as InnoDB.
- The time needed to perform the snapshot does not increase as the size of the database grows.
  - The backup window (from the perspective of the application) is almost zero.
  - Snapshots use a copy-on-write method to ensure they are almost instantaneous.
  - There is a small performance cost while the snapshot is active.

"Copy-on-write" is a technique that copies data blocks on disk when they change, and is similar to the method used by the InnoDB undo log and MVCC. When you perform a snapshot, LVM marks all blocks that subsequently change and records the snapshotted version of those blocks. The initial snapshot takes a little time, but all subsequent write operations incur a small performance and storage cost while the snapshot is active.

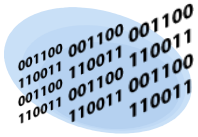
Replication-Based Backups

- MySQL supports one-way asynchronous replication, in which one server acts as the master while one or more other servers act as slaves.
- MySQL replication can be used for backups:
  - The master is used for the production applications.
  - A slave is used for backup purposes.
- This eliminates the impact on production applications.
- The backup of the slave is either logical or raw.
- Higher cost: There must be another server and storage to store the replica of the database.
- Based on asynchronous replication:
  - The slave may be delayed with respect to the master.
  - This is acceptable if the binary log is not purged before the slave has read it.

Binary Log Backups

Binary log backups record modifications to the data.

- Useful for minimizing exposure between full backups by restoring events that have occurred since the last full backup
- Advantages:
  - Binary log backups contain a record of all changes to the data over time.
    - Other backup types contain a snapshot of the data.
  - You can take multiple binary log backups in sequential order.



- Disadvantages:
  - You must sequentially restore all sequential binary logs that were created from the last full backup.
  - Recovery from a system failure can be slow depending on the number of binary logs that must be restored.

Binary Logging and Incremental Backups

- Control binary logging at the session level:

```
SET SQL_LOG_BIN = {0|1|ON|OFF}
```

  - You must have the `SUPER` privilege to set this variable.
- Flush binary logs when taking logical or raw backups.
  - Synchronize binary logs to backups.
- Logical and raw backups are full backups.
  - All rows of all tables are backed up.
- To perform an incremental backup, copy binary logs.
- Binary logs can be used for fine-grain restores.
  - You can identify transactions that caused damage and skip them during restore.

Quiz

Which backup type converts databases and tables to SQL statements?

a. Logical backup
b. Physical backup
c. Replication-based backup
d. Snapshot-based backup

Answer: a

Topics

- Understanding backups
- Backup techniques
- Creating a backup strategy

Comparing Backup Methods

Method	Hot/Warm/ Cold	Storage Engines	Logical/ Physical	Consistent	Availability
MySQL Enterprise Backup	Hot (InnoDB)/ warm (other)	All	Physical	Yes	Commercially available
mysqldump or **mysqlpump**	Hot (InnoDB)/ warm (other)	All	Logical	Yes	Freely available
Snapshots	Hot	All	Physical	Yes	Need snapshot volume or file system
Replication	Hot	All	Logical or physical	Yes	Freely available
SQL Statements	Warm	All	Logical	No	Freely available
OS Copy Commands	Cold or warm	All	Physical	Yes	Freely available

mysqldump and **mysqlpump**

These utilities can perform a hot backup on InnoDB tables only when the entire operation is wrapped in a transaction by specifying the `--single-transaction` option.

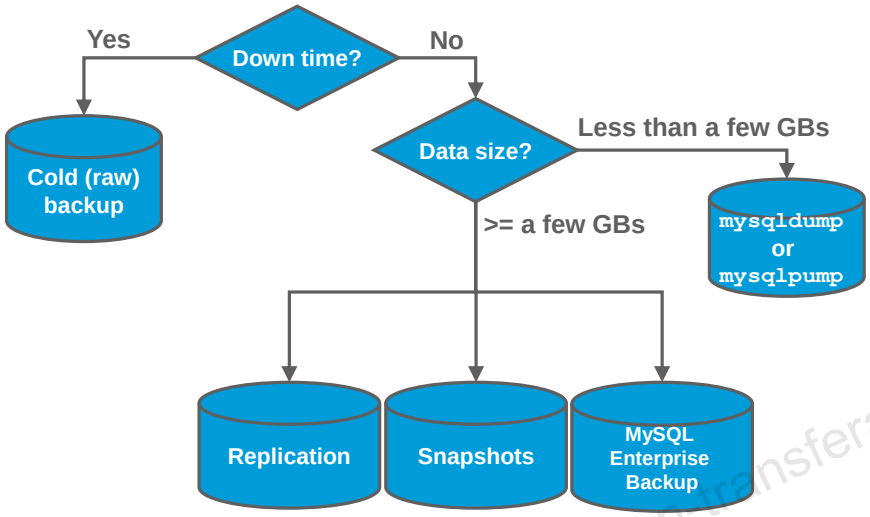
Snapshots

Snapshots do not work in the same way for all engines. For example, InnoDB tables do not require `FLUSH TABLES WITH READ LOCK` to start a snapshot, but MyISAM tables do.

Deciding a Backup Strategy

- The backup and recovery strategy that you implement is determined by:
  - How complete is the data record?
    - You might choose not to back up every table, or all data in each table.
  - How current is the backup?
    - Some data does not change very often, so you might choose to back up some tables (or portions of some tables) infrequently.
- Other questions to ask when deciding your strategy:
  - How much down time can your system afford, if any?
  - How much data is there to back up?
  - Which storage engines are being used to store the data (InnoDB, MyISAM, or both)?

Backup Strategy: Decision Chart



More Complex Strategies

Combine multiple backup techniques to create more complex strategies:

- Example using full and binary log backups:
  - Nightly backups using `mysqlbackup` from a replication slave
  - Multiple binary log backups each hour to minimize exposure during each day
- Example using partial backups:
  - Technique:
    - `mysqldump` with `--where` option
    - SELECT INTO OUTFILE
  - Weekly conditional backups of large transactional tables containing only fixed or historical data
    - Data that does not change, for example, fulfilled orders that have passed the “return by” date
  - Daily/hourly supplemental backups containing live data

If you can be sure that historical data before a threshold date does not change, you do not need to back it up more than once. Backing up only data after that point is faster, although it means you must take care that your backups represent a consistent view of the data, and the restore operation is more complex.

Summary

In this lesson, you should have learned how to:

- Distinguish between the different types of backups
- State the advantages and disadvantages of the various backup techniques
- Implement a backup strategy

Practices

- 10-1: Quiz – Backup Strategies

11

Performing Backups

Objectives

After completing this lesson, you should be able to:

- Use MySQL Enterprise Backup to perform consistent backups
- Use the `mysqldump` and `mysqlpump` utilities to perform logical backups
- Explain when and how to use raw file backups
- Back up the binary log

Topics

- MySQL backup tools
- Raw backup methods
- Techniques that use the binary log

Backup Tools: Overview

- SQL statements for logical backups
- SQL statements combined with operating system commands for raw backups
  - Example: `LOCK TABLES`
- Other backup tools for MySQL:
  - MySQL Enterprise Backup : Raw backups
  - **`mysqldump`**: Logical backups
  - **`mysqlpump`**: Logical backups
- Third-party tools

Oracle commercial and community tools are the primary focus of this lesson. There are commercial and community third-party tools that you can incorporate into your backup strategies.

MySQL Enterprise Backup

- From the command line: **`mysqlbackup`**
- Incremental backups:
  - Back up data that changed since the previous backup
  - Are primarily intended for InnoDB tables, or non-InnoDB tables that are read-only or rarely updated
- Single-file backups:
  - Provide the ability to create a backup in a single-file format
  - Can be streamed or piped to another process
    - Example: Tape backup or a command, such as `scp`
    - Put the backup on another storage device or file server without significant storage overhead on the original database server.

MySQL Enterprise Backup: Storage Engines

- InnoDB
  - MySQL Enterprise Backup performs **hot** backups on InnoDB tables.
    - The backup takes place while applications connected to the database are running.
  - This type of backup does not block normal database operations.
    - It captures even changes that occur while the backup is happening.
- Other storage engines
  - MySQL Enterprise Backup performs a **warm** backup.
    - The database tables can be read by applications.
    - The database cannot be modified while the non-InnoDB backup runs.

MySQL Enterprise Backup does not support offline (cold) backups.

MySQL Enterprise Backup: InnoDB Files

Raw InnoDB files that are backed up with `mysqlbackup`:

- `ibdata*` files: Shared tablespace files that contain the system tablespace and possibly the data for some user tables
- `.ibd` files: Per-table data files and general tablespace files
- `ib_logfile*` files
  - Data extracted from the `ib_logfile*` files is stored in a new backup file named `ibbackup_logfile`.
  - This includes redo log information representing changes that occur while the backup is running.

MySQL Enterprise Backup: Non-InnoDB Files

- MySQL Enterprise Backup is optimized for InnoDB.
- You can use MySQL Enterprise Backup to backup non-InnoDB data, if the following are true:
  - The MySQL server you want to back up supports InnoDB
  - The server contains at least one InnoDB table
- By default, `mysqlbackup` backs up all files in the data directory.
  - If you specify the `--only-known-file-types` option, the backup includes only additional files with MySQL-recognized extensions.
- For example, the following files are backed up for data stored with MyISAM
  - `.MYD`: MyISAM data files
  - `.MYI`: MyISAM index files
  - `.sdi`: Metadata files

MySQL Enterprise Backup: Use Cases

- Use MySQL Enterprise Backup to take:
  - An online (hot) backup of your InnoDB tables
  - A snapshot of your MyISAM tables that corresponds to the same binlog position as the InnoDB backup
- In addition to creating backups, MySQL Enterprise Backup can:
  - Pack and unpack backup data
  - Apply any changes to InnoDB tables that occurred during the backup operation to the backup data
  - Copy data, index, and log files back to their original locations

MySQL Enterprise Backup: Basic Usage

```
mysqlbackup --user=username --password --port=portnumber
      --backup_dir=backup-directory command
```

- Provide credentials and server coordinates using the same options as the `mysql` client:
  - Including `--user`, `--port`, and `--password`
- The `--backup_dir` option specifies the directory to store the backup data and metadata.
- Basic commands include:
  - **`backup`**: Performs the initial phase of a backup
  - **`backup-and-apply-log`**: Includes the initial phase of the backup and a second phase that brings the InnoDB tables in the backup up-to-date, including any changes made to the data while the backup was running

Backup Process

1. `mysqlbackup` opens a connection to the MySQL server.
2. `mysqlbackup` takes an online backup of InnoDB tables.
   - This phase does not disturb normal database processing.
3. When the `mysqlbackup` run has almost completed, it:
   - Executes the SQL command `FLUSH TABLES WITH READ LOCK`
   - Copies the non-InnoDB files (such as MyISAM `.MYI` and `.MYD` files) to the backup directory.
4. `mysqlbackup` runs to completion and `UNLOCKs` the tables.

The read-locked phase lasts only a short time if:

- You do not run long `SELECT` or other queries in the database while the read lock is in effect
- Your MyISAM tables are small

Otherwise, the whole database, including InnoDB type tables, are locked until all long-running queries that started before the backup have completed.

Restoring a Backup with MySQL Enterprise Backup

Basic usage:

```
mysqlbackup --backup-dir=backup-dir  copy-back
```

- **backup-dir**: Specifies where the backup files are stored
- **copy-back**: Instructs MySQL Enterprise Backup to perform a restore operation

The restore operation copies the contents of *backup-dir* to their original locations.

Using the `copy-back` Command

- Shut down the database server before using the `copy-back` command.
- When you use the `copy-back` command, MySQL Enterprise Backup:
  - Copies the data files, logs, and other backed-up files from the backup directory to their original locations
  - Performs any required post-processing on them
- During the `copy-back` process, `mysqlbackup` cannot query its settings from the server, so it reads the standard configuration files for options such as the `datadir`.
  - If you want to restore to a different server, you can provide a non-standard defaults file with the `--defaults-file` option.

MySQL Enterprise Backup: Single-File Backups

- Basic usage:

```
mysqlbackup -uuser -ppassword
--backup-image=image-file
--backup_dir=backup-dir backup-to-image
```

- Other scenarios
  - Standard output:

```
mysqlbackup -uuser -ppassword
--backup-image=image-file
--backup_dir=backup-dir
--backup-image=- backup-to-image > image-file
```

  - Convert an existing backup directory to a single file:

```
mysqlbackup
--backup-image=image-file
--backup_dir=backup-dir
--backup-image=image-file backup-dir-to-image
```


In many UNIX commands, the dash symbol "–" represents standard output. When you use it in the option `--backup-image=-`, it sends the backup to standard output, from which you can redirect it with standard output redirection as with `> image-file` in the example in the slide.

MySQL Enterprise Backup: Restoring Single-File Backups

- Extract a selection of files:

```
mysqlbackup
--backup-image=image-file
--backup_dir=backup-dir image-to-backup-dir
```

- Other scenarios
  - List contents:

```
mysqlbackup
--backup-image=image-file list-image
```

  - Convert an existing backup directory to a single file:

```
mysqlbackup
--backup-image=image-file
--src-entry=file-to-extract
--dst-entry=file-to-extract extract
```


- **`--src-entry`:** Identifies a file or directory to extract from a single-file backup
- **`--dst-entry`:** Is used with single-file backups to extract a single file or directory to a user-specified path

Basic Privileges Required for MySQL Enterprise Backup

- Backup operations require certain privileges.
  - Use an existing MySQL `root` account.
  - Or, create a new account with sufficient privileges.
- Minimum privileges required:
  - `RELOAD` on all databases and tables
  - `CREATE`, `INSERT`, `UPDATE`, and `DROP` on tables:
    - `mysql.backup_progress`
    - `mysql.backup_history`
  - `SELECT` and `ALTER` on `mysql.backup_history` table
  - `SUPER`, to enable and disable logging, and to optimize locking in order to minimize disruption to database processing
  - `REPLICATION CLIENT` to retrieve the binlog position stored with the backup
  - `PROCESS` to process DDL statements with `ALGORITHM=INPLACE` clause

The `mysqlbackup` client connects to the server with the `--user` and `--password` options. Specifying login details at the command line is not ideal, so consider including them in the [client] section of the `mysqlbackup` `my.cnf` file (or other configuration file). `mysql_config_editor` can be used to create an encrypted password in the `.mylogin.cnf` file.

Refer to the documentation for other privileges required in some special scenarios when certain features are used.

Granting Required Privileges

The following script sets the required permissions for the `backupuser` user:

```
CREATE USER 'backupuser'@'localhost' IDENTIFIED BY 'new-password';
GRANT RELOAD ON *.* TO 'backupuser'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE ON mysql.backup_progress TO
    'backupuser'@'localhost';
GRANT CREATE, INSERT, DROP, UPDATE, SELECT, ALTER ON mysql.backup_history TO
    'backupuser'@'localhost';
GRANT REPLICATION CLIENT ON *.* TO 'backupuser'@'localhost';
GRANT SUPER ON *.* TO 'backupuser'@'localhost';
GRANT PROCESS ON *.* TO 'backupuser'@'localhost';
```

Quiz

Which raw InnoDB files are backed up with the `mysqlbackup` command?

a. `ibdata*` files

b. `.ibd` files

c. `ib_logfile*` files

d. All of the above

Answer: d

`mysqldump` and `mysqlpump`

- The `mysqldump` and `mysqlpump` utilities are included in the MySQL distribution.
- They perform logical backups and work with any database engine.
  - Certain features such as “hot” backups can be performed only on InnoDB tables.
- They can be automated by using `crontab` in Linux and UNIX, and the Windows Task Scheduler in Windows.
- There are no tracking or reporting tools for `mysqldump` or `mysqlpump`.

mysqldump

- Dumps table contents to files:
  - All databases, specific databases, or specific tables
  - Allows backup of local or remote servers
  - Independent of the storage engine
  - Written in text format
  - Portable
  - Excellent copy/move strategy
  - Good for small exports but not a full backup solution
- Basic usage:

```
mysqldump --user=username --password=password db_name > backup.file
```

Storage Location

For SQL-format dump files that contain `CREATE TABLE` and `INSERT` statements for re-creating the tables, the server sends the table contents to `mysqldump`, which writes the files on the client host.

Ensuring Data Consistency with `mysqldump`

Ensuring consistency:

- **`--master-data`** option alone
  - Locks tables during backup
  - Records the binlog position as a `CHANGE MASTER TO` statement in the backup file
    - `--master-data=2` records the position as a comment
- **`--master-data`** and **`--single-transaction`** options used together
  - Does not lock tables – only InnoDB table consistency is guaranteed
  - Acquires a global lock at the beginning of the backup operation to obtain a consistent binary log position
- **`--lock-all-tables`**
  - Satisfies consistency by locking tables
- **`--flush-logs`**
  - Starts a new binary log

`mysqldump` Options for Creating Objects

- **`--no-create-db`**
  - Suppresses the `CREATE DATABASE` statements
- **`--no-create-info`**
  - Suppresses the `CREATE TABLE` statements
- **`--no-data`**
  - Creates the database and table structures but does not dump the data
- **`--no-tablespaces`**
  - Tells the MySQL server not to write any `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statements to the output
- **`--quick`**
  - Retrieves single rows from a table, without buffering sets of rows

`mysqldump` Options for Dropping Objects

- **`--add-drop-database`**
  - Adds a `DROP DATABASE` statement before each `CREATE DATABASE` statement
- **`--add-drop-table`**
  - Adds a `DROP TABLE` statement before each `CREATE TABLE` statement

mysqldump General Options

- MySQL stored routines, procedures, and triggers:
  - **--routines**
    - Dumps stored routines (procedures and functions) from the dumped databases
  - **--triggers**
    - Dumps triggers for each dumped table
- Top options in one option (**--opt**)
  - Shorthand for the most common options to create an efficient and complete backup file
    - Includes the --add-drop-table, --add-locks, --create-options, --disable-keys, --extended-insert, --lock-tables, --quick, and --set-charset options.

MySQL 8.0 stores stored routines and scheduler events in the data dictionary instead of tables as in previous versions of MySQL server. Therefore, a backup of the mysql database does not contain the stored procedures and events. You must explicitly specify the --routines and --events options in mysqldump to back them up.

The --triggers and --opt options are enabled by default; use --skip-triggers and --skip-opt to disable them.

Restoring `mysqldump` Backups

- Reload `mysqldump` backups with `mysql`:

```
mysql --login-path=login-path database < backup_file.sql
```

  - You must name the database if the backup file does not contain the `USE` statements that specify the database.
  - If you run `mysqldump` with `--database` or `--all-databases` options:
    - The dump file contains appropriate `USE db_name` statements.
    - You do not need to specify the target database name when reloading from that dump file.

- Copy from one database to another:

```
mysqldump -uuser -ppassword
orig-db table | mysql --login-path=login-path copy-db
```


Copying from One Database to Another

You can use `mysqldump` output to restore tables or databases and to copy them. `mysql` can read from a pipe, so the use of `mysqldump` and `mysql` can be combined into a single command that copies tables from one database to another. The pipe technique also can be used to copy databases or tables over the network to another server.

Using `mysqlimport`

- If you invoke `mysqldump` with the `--tab` option, it produces tab-delimited data files:
  - A **`.sql`** file that contains `CREATE TABLE` statements
  - A **`.txt`** text file that contains the table data
- To reload the table:
  1. Change to the backup directory:
  2. Process the `.sql` file by using `mysql`.
  3. Load the `.txt` file by using `mysqlimport`.

```
1 cd backup_dir

2 mysql --login-path=login-path database < table.sql

3 mysqlimport -uuser -ppassword database table.txt
```


`mysqlimport`

If you combine the `--tab` option with options such as `--fields-terminated-by` and `--fields-enclosed-by`, which perform format control, specify the same format-control options with `mysqlimport` so that it knows how to interpret the data files.

Privileges Required for `mysqldump`

You must have the following privileges to use `mysqldump`:

- `SELECT` for dumped tables
- `SHOW VIEW` for dumped views
- `TRIGGER` for dumped triggers
- `LOCK TABLES` (unless you use the `--single-transaction` option)
- Other options might require extra privileges. For example:
  - To use the `--flush-logs` or `--master-data` options, you must have the `RELOAD` privilege.
  - To create a tab-delimited output with the `--tab` option, you must have the `FILE` privilege.
  - To use the `--routines` option to back up stored functions and procedures, you must have the global `SELECT` privilege.

Privileges Required for Reloading Dump Files

To reload a dump file, you must have the following privileges:

- `CREATE` on each of the dumped objects
- `ALTER` on the database, if the `mysqldump` output includes statements that change the database collation to preserve character encodings

mysqlpump

`mysqlpump` is very similar to `mysqldump`, but with the following enhancements:

- Provides better performance than `mysqldump` by extracting data in parallel threads
  - `mysqlpump` divides the dump process into several subtasks and then adds these subtasks to a multi-threaded queue.
  - This queue is then processed by *N* threads (two by default).
- Enables better control over which database objects to dump on each thread
- Dumps users as `CREATE USER/GRANT` statements instead of as `INSERT`s into the `mysql` system database
- Enables compressed output
- Displays a progress indicator
- Provides faster secondary index reloading for InnoDB tables

Faster secondary index reloading for InnoDB tables is achieved by adding indexes after the rows are inserted.

Specifying Objects to Back Up with `mysqlpump`

The `mysqlpump` utility provides many options for specifying which database objects to back up, for example:

- Back up all databases (default):

```
mysqlpump -uuser -ppassword > full_backup.sql
```

- Back up *only* the `employees` and `world` databases:

```
mysqlpump -uuser -ppassword
--databases employees world > emp_world_backup.sql
```

- Dump all databases with names that start with "db."

```
mysqlpump -uuser -ppassword
 --include-databases=db% --result-file=all_db_backup.sql
```

- Back up everything *except* tables named `t1`.

```
mysqlpump -uuser -ppassword --exclude-tables=t1
--result-file=partial_backup.sql
```

ORACLE

Unlike `mysqldump`, the default behavior of `mysqlpump` is to back up *all* databases. This option is implicit if you execute `mysqlpump` without specifying which schemas to include.

By default, `mysqlpump` does not dump the following internal databases: `PERFORMANCE_SCHEMA`, `INFORMATION_SCHEMA`, `sys`, and `ndbinfo`.

`mysqlpump` dumps user accounts in logical form using `CREATE USER` and `GRANT` statements (for example, when you use the `--include-users` or `--users` option). For this reason, dumps of the `mysql` system database do not by default include the grant tables that contain user definitions: `user`, `db`, `tables_priv`, `columns_priv`, `procs_priv`, or `proxies_priv`. To dump any of the grant tables, name the `mysql` database followed by the table names.

Parallel Processing with `mysqlpump`

You can configure the number of threads `mysqlpump` uses with the **`--default-parallelism`** and **`--parallel-schemas`** options. For example:

- Back up all databases with four threads:

```
mysqlpump --uuser -ppassword
--default-parallelism=4 > full_backup.sql
```

- Create two queues: one to process `db1` and `db2` and the other to process `db3` and `db4`. Use five threads on the first queue, two on the second, and three on the default queue for all other schemas:

```
mysqlpump -uuser -ppassword
 --parallel-schemas=5:db1,db2
 --parallel-schemas=2:db3,db4
 --default-parallelism=3 > full_backup.sql
```

Note

The default number of threads (the value of `--default-parallelism`) is two.

Quiz

`mysqldump` and `mysqlpump` are useful for small exports but not as a full backup solution.

a. True
b. False

Answer: a

Topics

- MySQL backup tools
- Raw backup methods
- Techniques that use the binary log

Raw InnoDB Backups: Overview

- A raw (physical) backup operation makes a complete InnoDB backup.
  - Contains a backup of all InnoDB tables
  - Makes exact copies of all tablespace files
- All InnoDB tables in all databases must be backed up together.
  - InnoDB maintains some information centrally in the system tablespace.
  - Other InnoDB tablespaces contain table data that is dependent on InnoDB's data dictionary in the system tablespace.
- You must shut down the MySQL server for the duration of the file copy to ensure consistency across all tablespaces.
  - This is a "cold" backup.

Portability of Raw Backups

- Raw (binary) portability
  - Raw (binary) database files can be copied from one MySQL server to another.
  - Binary portability is useful when taking a binary backup made on one machine to use on a second machine that has a different architecture.
    - For example, you can copy databases from one MySQL server to another by copying the raw database files.
- Storage engine portability
  - InnoDB:
    - All tablespace and log files for the database can be copied directly.
      - Example: You can copy tablespace files directly from a MySQL server on one machine to a MySQL server on another machine, and the second server accesses the tablespace.
    - The database directory name must be the same on the source and destination systems.
  - MyISAM and Archive:
    - All files for an individual table can be directly copied

File name compatibility for Windows:

- The MySQL server internally stores lowercase database and table names on Windows systems.
- For case-sensitive file systems, use an option file statement:

```
lower_case_table_names=1
```

Raw InnoDB Backup Procedure

1. Perform a slow (clean) shutdown of the server.
   - Requires `innodb_fast_shutdown=0` (The default value is `1`.)
   - Allows additional InnoDB flushing operations to complete before shutdown
   - Shutdown takes longer; startup is faster.
2. Make a copy of *all* InnoDB data, log, and configuration files:
   - Data files: `ibdata` and `*.ibd`
   - Log files: `ib_logfile`
   - All configuration files that the server applies, such as `my.cnf`
3. Restart the server.

Recovering from Raw InnoDB Backups

- To recover InnoDB tables by using a raw backup:
  1. Stop the server.
  2. Replace all the components of which copies were made during the backup procedure.
  3. Restart the server.
- InnoDB stores table metadata in the shared tablespace. You must either:
  - Copy the shared tablespace and per-table tablespace files as a group.
    - This replaces the target system tablespace.
  - Import exported per-table tablespaces individually by using **`ALTER TABLE ... IMPORT TABLESPACE`**.

Using Transportable Tablespaces for Backup

When you back up InnoDB tables by using transportable tablespaces:

- You can perform partial backups.
  - Per-table backups, or multiple tables within a business function
- You do not need to load those tables in another server.
  - If you want to read the backed-up copies for analytics or other purposes, ensure InnoDB is set to read-only on that server.
  - You must also ensure that the original server and the read-only server have matching InnoDB page sizes and row formats.
- You must not change the tables by using them in another server.
  - If you change the tables, the backup no longer has integrity.
- You can restore partial backups per table or function.

Transportable Tablespaces: Copying a Table to Another Instance

1. On the destination instance, create a table with the same structure as that on the source instance.
2. On the destination instance, discard the existing tablespace:

```
mysql> ALTER TABLE db.table DISCARD TABLESPACE;
```

3. On the source instance, execute **`FLUSH TABLES...FOR EXPORT`** to quiesce the table and create the `.cfg` metadata file:

```
mysql> FLUSH TABLES db.table FOR EXPORT;
```

   – The `.cfg` file is created in the InnoDB data directory.

4. Copy the `.ibd` and `.cfg` file from the source instance to the destination instance.
5. On the source instance, release the locks by executing `UNLOCK TABLES`.
6. On the destination instance, import the tablespace:

```
mysql> ALTER TABLE db.table IMPORT TABLESPACE;
```

ORACLE

If you are copying an InnoDB partitioned table, then there will be a separate tablespace (`.ibd` file) and `.cfg` file for each partition. You must discard each of the tablespaces on the destination instance, copy all of the `.ibd` and `.cfg` files and import all the tablespaces.

Raw MyISAM and ARCHIVE Backups

- Raw MyISAM and ARCHIVE backups comprise the files that MySQL uses to represent the tables.
  - For MyISAM, these are the `.sdi`, `.MYD`, and `.MYI` files.
  - For ARCHIVE tables, these are the `.sdi` and `.ARZ` files.
- During this copy operation, other programs (including the server) must not modify the files being copied.
  - To avoid server interaction problems, lock and flush the tables or stop the server during the copy operation.

Note

Locking tables instead of shutting down the server works on Linux systems. On Windows, file-locking behavior is such that it might not be able to copy table files for tables that are locked by the server. In that case, stop the server before copying table files.

Raw MyISAM and ARCHIVE Backup Procedure

1. While the server is running, lock the table to be copied:

```
mysql> USE world_myisam
mysql> FLUSH TABLES city WITH READ LOCK;
```

2. Perform a file system copy.
3. Start a new binary log file:

```
mysql> FLUSH LOGS;
```

   – The new binary log file contains all statements that change data after the backup (and any subsequent binary log files).

4. Release the lock after the file system copy:

```
mysql> UNLOCK TABLES;
```


If you do not want to flush the logs, you can record the current log coordinates with the `SHOW MASTER STATUS` command, which returns the current binary log file name and position.

Recovering from Raw MyISAM or Archive Backups

To recover a MyISAM or ARCHIVE table from a raw backup:

1. Drop the existing table if it exists.
2. Copy the backup data file into the appropriate database directory.
3. Copy the backup `.sdi` file into a temporary directory.
4. Run the `IMPORT TABLE` statement specifying the path to the `.sdi` file to load the table into the database.

Note that the `.sdi` file contains the schema name and table name. You may need to edit the file if you are restoring it to a different database or table name.

The `IMPORT TABLE` statement loads the data dictionary into the MySQL server.

LVM Snapshots

Logical Volume Manager (LVM):

- Manages storage areas on physical media that are more flexible than disk partitions
  - Physical volumes
  - Logical volumes
  - Volume groups
- Uses a mechanism known as “copy-on-write” to create snapshots
  - The snapshot behaves as if it is an instantaneous copy of the file system at the time you create the snapshot.
  - Internally, the snapshot records the original versions of disk blocks that change after you create the snapshot.
  - When you access the snapshot, LVM returns either unchanged blocks from the file system, or the original copy of changed blocks.

Copy-On-Write Snapshots

When you read files from a newly created snapshot, LVM reads those files from the original volume. As the original volume changes, LVM copies data to the snapshot immediately before it changes on the original, so that any data that has changed since taking the snapshot is stored in its original form in the snapshot. The result is that when you read files from the snapshot, it delivers the version of data existing at the instant the snapshot was created.

Because a snapshot is almost instantaneous, you can assume that no changes to the underlying data take place during the snapshot. This makes snapshots very useful for backing up InnoDB databases without shutting the server down.

Creating LVM Snapshots

- To create a snapshot, use the following syntax:

```
lvcreate -s -n snapshot-name -L size original-volume
```

  - `-s`: Instructs `lvcreate` to create a snapshot
  - `-n` *snapshot-name*: The new snapshot's name
  - `-L` size: The new snapshot's size
  - *`original-volume`*: The original volume's location
- If you need the snapshot only for a short while, the reserved size can be much less than the size of the original volume.
  - The snapshot's storage contains only those blocks that change in the original.
  - For example, if you use the snapshot to perform a backup, it stores only those changes made during the time it takes to perform the backup and drop the snapshot.

Performance Implications

Over time, the snapshot's space requirement tends to grow to the size of the original volume. Also, each initial data change to blocks on the original volume causes two data writes to the volume group: the requested change and the copy-on-write to the snapshot. This can affect performance while the snapshot remains. For these reasons, you should remove the snapshot as soon as you have performed the backup.

LVM Backup Procedure

- Perform raw backups using LVM snapshots when:
  - The host supports LVM
    - Example: Linux supports LVM2.
  - The file system containing the MySQL data directory is on a logical volume
- Backup procedure:
  1. Take a snapshot of the logical volume containing MySQL's data directory.
     - Use `FLUSH TABLES WITH READ LOCK` if you are backing up non-InnoDB tables.
  2. Perform a raw backup from the snapshot.
  3. Remove the snapshot.

LVM Backup: Example

Assuming a volume group `VG_MYSQL` and a logical volume `lv_datadir`:

1. Create the snapshot:

```
lvcreate -s -n lv_datadirbackup -L 2G /dev/VG_MYSQL/lv_datadir
```

   – This creates a snapshot called `lv_datadirbackup` with a reserved size of 2 GB.

2. Mount the snapshot as if it were a standard volume.
3. Perform a raw backup from that volume as with any other raw backup (for example, by using `tar` or `cp`).
4. Remove the snapshot:

```
lvremove VG_MYSQL/lv_datadirbackup
```


Because you are backing up from a snapshot, the snapshot's copy of the database does not change during the backup process. You are sure to get a consistent version of the data files as they existed at the time of the backup, without needing to stop the server.

Backing Up Log and Status Files

- Binary log files
- Option files used by the server (`my.cnf` and `my.ini` files)
- Replication files:
  - `master.info`
  - `relay-log.info`
- Replication slave data files
  - `SQL_LOAD-*`
- MySQL binaries and libraries
- Strategies:
  - Static files: Backed up with normal system tools with the server running
  - Dynamic files: Backed up with normal system tools with the server stopped

Binary Log Files

Binary logs store updates that have been made after the backup was made.

Option Files Used by the Server (`my.cnf` and `my.ini` files)

These files contain configuration information that must be restored after a crash.

Replication Files

Replication slave servers create a `master.info` file that contains information needed for connecting to the master server, and a `relay-log.info` file that indicates the current progress in processing the relay logs.

Replication Slave Data Files

Replication slaves create data files for processing `LOAD DATA INFILE` statements. These files are located in the directory named by the `slave_load_tmpdir` system variable, which you can set by starting the server with the `--slave-load-tmpdir` option. If you do not set `slave_load_tmpdir`, the value of the `tmpdir` system variable applies. To safeguard replication slave data, back up the files beginning with `SQL_LOAD-`.

Topics

- MySQL backup tools
- Raw backup methods
- Techniques that use the binary log

Replication as an Aid to Backup

Replicate to another server to avoid backup overhead.

- Use the slave server to make backups instead of backing up from the master.
- By using a slave server for backups:
  - Applications that use the master are not slowed down by table locks or flushes.
  - The backup procedure does not add processing load on the master.
  - Backup files do not require additional hard disk space or processing on the master.

Backing Up from a Replication Slave

1. Stop the slave to make a backup:
   - `STOP SLAVE SQL_THREAD`
     - Flush the tables if you want to make a logical backup.
   - Alternatively, stop the server to make a raw file copy.
2. Back up the slave's databases:
   - Use `mysqldump` or `mysqlbackup` to back up the contents of a running server.
   - Use system tools to copy raw files if you stop the server.
3. Start the slave:
   - Start the server if it is stopped.
   - `START SLAVE SQL_THREAD`

Backing Up from Multiple Sources to a Single Server

- Multi-source replication can be used to:
  - Back up multiple servers to a single server
  - Merge table shards
  - Consolidate data from multiple servers to a single server
- The slave creates a replication channel for each master from which it receives transactions.
  - Use the **CHANGE MASTER TO ... FOR CHANNEL *channelname*** syntax:

```
CHANGE MASTER TO ...
  MASTER_LOG_FILE='binlog.000006',
  MASTER_LOG_POS=143 FOR CHANNEL 'shard-1';
```

- Replicate from all channels concurrently, or start and stop individual channels:

```
START SLAVE IO_THREAD FOR CHANNEL 'shard-1';
```


If you are consolidating multiple shards as part of a backup strategy, you might enable the slave I/O threads only at certain times of the day to avoid placing undue stress on the shard master servers during peak times. However, bear in mind that all relevant events on the master server must replicate, and that might take quite a lot of time at the slave server.

Processing Binary Log Contents

1. Determine which logs were written after a backup was made.
2. Convert the contents with **`mysqlbinlog`**:

```
mysqlbinlog binlog.000050 binlog.000051 binlog.000052 | mysql
```

   - Process all binlogs with one command.

3. Remove the binary logs that are no longer useful and that take up space on the server host:
   - Use the **`--delete-master-logs`** option when you create a backup using `mysqldump`.
   - Execute **`PURGE BINARY LOGS`** to remove files or events that are no longer useful.
     - Do not remove log files or events that have not yet been replicated or backed up.

```
PURGE BINARY LOGS TO 'binlog.000048';
```

After you restore the binary backup files or reload the text backup files, finish the recovery operations by reprocessing the data changes that are recorded in the server's binary logs.
To do this, you must determine which logs were written after the backup was made. The contents of these binary logs then need to be converted to text SQL statements with the `mysqlbinlog` program to process the resulting statements with `mysql`.

Selective Binary Log Processing

- Restore partial binlogs:
  - `--start-datetime` / `--stop-datetime`
  - `--start-position` / `--stop-position`

```
mysqlbinlog --start-position=23456 binlog.000004 | mysql
```

- Select only those events that occur after selecting a specific default database with `--database`:

```
mysqlbinlog --database=sales binlog.000043 | mysql
```

- Restore to a different database with `--rewrite-db`:

```
mysqlbinlog --rewrite-db='sales->sales_03'
  --stop-datetime='2018-03-31 23:59:59.999999' binlog.000082 | mysql
```

Point-in-Time Recovery

You might want to apply only portions of the binary log *before* or *after* a point in time:

- **Before**:
  - If you know that somebody accidentally executed a destructive statement and you want to restore changes up to that point
  - If you are restoring changes up to the last moment of a particular day to an analytics server
- **After**:
  - If you are using a binary log to apply changes after a full backup or after a flushed binary log that you have backed up, you can apply the log after that point.

Using `mysqlbinlog` for Point-in-Time Recovery

`mysqlbinlog` supports the following options:

- **`--start-datetime`**:
  - Specifies the date and time at which to begin extraction
  - Is given in `DATETIME` format
    - Note that the granularity of `--start-datetime` is only one second, so it might not be accurate enough to specify the exact position to start with.
- **`--start-position`**:
  - Specifies the log position of the first logged statement that you want to extract
- There are also corresponding **`--stop-datetime`** and **`--stop-position`** options for specifying the point at which to stop extracting log contents.

If you are not sure about the time stamp or position in a log file that corresponds to the point at which processing begins, use `mysqlbinlog` without `mysql` to display the log contents for examination:

```
mysqlbinlog file_name | more
```

Configuring MySQL for Restore Operations

- MySQL client programs might fail due to timeouts or buffer overflows.
  - During logical backup using `mysqldump`
  - During binary log restore operations using `mysqlbinlog`
- Configure the following variables:
  - `--max-allowed-packet=256M`
    - Default value: `64M`
    - Ensure that the value is large enough during backup and restore operations so that you do not exceed MySQL's packet size.
  - Ensure that the `--wait-timeout` value is not too small.
    - Default value: `28800` (8 hours)
    - If you set this to a much shorter value, the `mysqldump` or `mysqlbinlog` operations might fail.

Quiz

Which command converts the contents of binary logs?

a. `binconvert`
b. `mysql`
c. `mysqlbinlog`
d. `mysqldump`

Answer: c

Summary

In this lesson, you should have learned how to:

- Use MySQL Enterprise Backup to perform consistent backups
- Use the `mysqldump` and `mysqlpump` utilities to perform logical backups
- Explain when and how to use raw file backups
- Back up the binary log

Practices

- 11-1: Backing up with MySQL Enterprise Backup
- 11-2: Backing up with `mysqldump` and `mysqlpump`
- 11-3: Backing up by Using the Binary Log

Configuring a Replication Topology

12

MySQL

ORACLE

Objectives

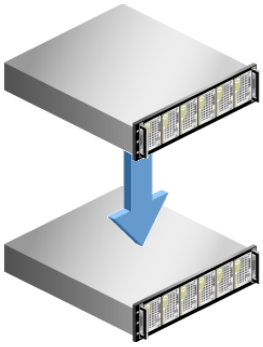
After completing this lesson, you should be able to:

- Describe MySQL replication
- Explain the role of replication in high availability and scalability
- Set up a MySQL replication environment
- Design advanced replication topologies

Topics

- Replication overview
- Replication conflicts
- When to use replication
- Configuring replication

MySQL Replication



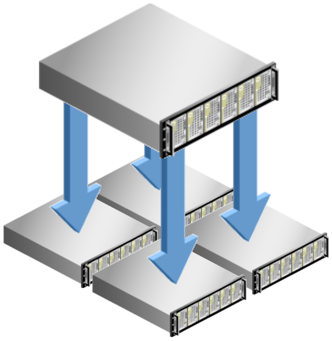
Replication is a feature of MySQL that enables servers to copy changes from one instance to another.

- The ***master*** records all data and structural changes to the binary log.
  - The binary log format is statement based, row based, or mixed.
- The ***slave*** requests the binary log from the master and applies its contents locally.
  - It records the status of all the received and applied events so that it can resume from where it has stopped after a server restart or network failure.

Note

- **Binary log formats:** MySQL supports statement-based, row-based, and mixed format logging as described in the “Binary Log Formats” slide later in the lesson.
- **Network Outages:** Replication in MySQL survives a network outage. Each slave keeps track of how much of the log it has processed, and resumes processing automatically when network connectivity is restored. This behavior is automatic and requires no special configuration.

Replication Masters and Slaves



The master/slave relationship is one-to-many:

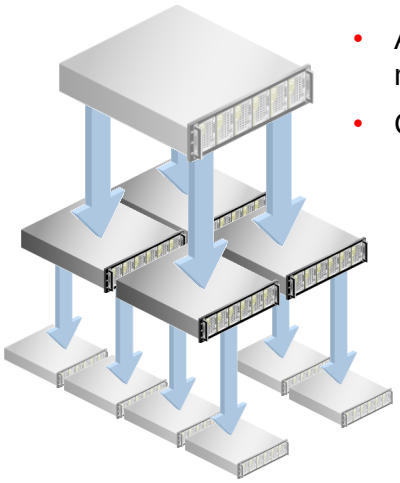
- Each slave reads logs from one master.
- A master can ship logs to many slaves.

Number of Slaves

There is no limit on how many slaves a single master can have. However, each additional slave uses a small amount of resources on the master, so you should carefully consider the benefit of each slave in production setups. The optimal number of slaves for a master in a given environment depends on several factors: size of schema, number of writes, relative performance of master and slaves, and factors such as CPU and memory availability. A general guideline is to limit the number of slaves per master to no more than 30.

Note: Usually a slave replicates only from a single master. However, a slave can replicate from multiple master servers in a Multisource replication scenario. Configuring Multisource replication is described in the slide titled "Multisource Replication."

Relay Slaves

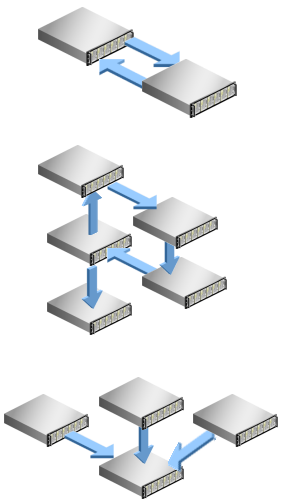


- A relay slave is a replication slave that acts as a replication master to another slave.
- Changes propagate to further slaves.

Relay Slave

A slave can act as a master to another slave. Changes that occur at the top-most master are requested and applied by its immediate slaves, which relay the changes down to their slaves, and so on until replication reaches the end of the chain. This enables updates to propagate through multiple levels of replication, allowing for topologies that are more complex. Each additional level adds more propagation delays into the system, so a shallower setup suffers from less replication lag than a deeper one.

Complex Topologies



More complex topologies are possible:

- A *bi-directional* topology has two master servers, each a slave of the other.
- A *circular* topology has any number of servers.
  - Each is both a master and a slave of another master.
  - Changes on any master replicate to all servers.
  - Not every slave must be a master.
- *Multisource* replication allows one slave to receive transactions from more than one master.

Note

Standard MySQL replication does not perform conflict resolution.

Quiz

A *relay slave* is:

a. A server that logs any transactions that cannot be applied to other slaves
b. A server that resolves conflicts by coordinating changes from multiple masters
c. A slave server that acts as a master to another slave
d. The first slave in a circular replication topology

Answer: c

Topics

- Replication overview
- Replication conflicts
- When to use replication
- Configuring replication

Replication Conflicts

Conflicts are possible in replication topologies with multiple masters.

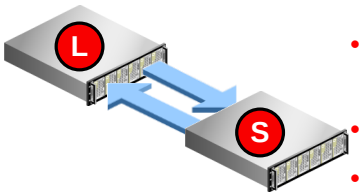
- If two clients write to the same row on two masters at approximately the same time, you cannot predict the row's final value at slave servers.
- The final value depends on the event ordering at the relay slave.
  - In hierarchical replication, the row's final value on slaves is implied by the hierarchy:
    - Slaves do not have the same value as masters if some intermediate master changed the row.
  - In circular replication, the row's final value is inconsistent across servers in case of a conflict.
    - The value depends on the order in which each master applies the event.

Conflict Resolution

In a typical configuration, clients write changes only to the master but read changes from any server. In an environment where servers allow concurrent updates on similar data, the eventual state of the data on multiple servers might become inconsistent. It is the responsibility of the application to prevent or manage conflicting operations. MySQL replication does not perform conflict resolution.

Conflicts can occur in any topology that includes more than one master. This includes simple hierarchies such as that shown in the slide titled "Relay Slaves," if a relay slave accepts changes from clients. Conflicts are particularly prevalent in circular topologies.

Replication Conflicts: Example Scenario with No Conflict

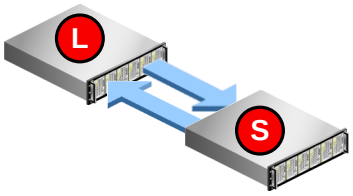


- "Luxury Brands" increases the price of luxury products by 20%. L
- "Special Events" reduces the prices of products over $500 by $50. S
- One luxury product costs $520 before these changes.
- If the operations perform in the order L and then S, the final result is $574 on both servers.
- If the operations perform in the order S and then L, the final result is $564 on both servers.

For example, consider an e-commerce company that implements replication using a circular topology, in which two of the servers deal with applications in the "Luxury Brands" and "Special Events" teams, respectively. Assume that the applications do not manage conflicting operations, and the following events occur:

- The "Luxury Brands" team increases the price of luxury products by 20%.
- The "Special Events" team reduces the prices of all products over $500 by $50 because of an upcoming special holiday.
- One product costing $520 falls into both categories and has its value updated by both of the preceding operations.

Replication Conflicts: Example Scenario with Conflict



- If both operations occur at approximately the same time, the following operations occur:
  - L Increases the product's price by 20%, from $520 to $624
  - S Reduces the product's price by $50, from $520 to $470
- If these operations occur at the same time, the changes each replicate to the other server resulting in a conflict.
  - The "Luxury Brands" server assumes a final value for that product of $574. L
  - The "Special Events" server assumes a final value of $564. S

The eventual price of the product on each server depends on the order in which each server performs the operations.

- Other servers in the replication environment assume final values depending on the order in which they applied the operations.

Similarly, conflicts occur if the "Luxury Brands" team adds a new product that has not replicated fully by the time the "Special Events" team makes its changes, or if two teams add a product with the same primary key on different servers.

As MySQL servers cannot detect and resolve replication conflicts, the application has to handle all the replication conflicts that may occur.

Note for MySQL Cluster users: MySQL Cluster uses a form of replication internally that differs in some ways from replication in MySQL Server, and offers conflict detection (and optional resolution).

Topics

- Replication overview
- Replication conflicts
- When to use replication
- Configuring replication

Replication Use Cases

Common uses for replication:

- **Horizontal scale-out**: Distribute the querying workload across multiple slaves.
- **Business intelligence and analytics**: Run expensive reports and analytics on a slave, letting the master focus on production applications.
- **Geographic data distribution**: Serve local users with a local application, and replicate business intelligence data to corporate servers.
- **High availability**: Provide redundancy between multiple servers, and facilitate controlled switchovers or rolling upgrades.

Replication for Horizontal Scale-Out

- Distribute the client query load across multiple servers:
  - Clients send write operations to the replication master.
  - Clients distribute read operations between replication slaves.
    - Use a load balancer or connectors that provide load balancing.
- Conflicts cannot occur because only one server accepts write operations.
  - Inconsistencies are possible if an application writes a value and immediately tries to read it again.
- Scale-out is not transparent. You must modify clients to:
  - Write to the master
  - Read from a load-balanced set of slaves
  - Handle potential inconsistencies that occur due to the delay in replicating from master to slaves

Horizontal scale-out: The most popular reason to implement replication is to distribute the querying workload across one or more slave servers to improve the performance of read operations throughout the application, and write operations on the master server by reducing its read workload.

Replication for Business Intelligence and Analytics

- Business intelligence workloads often have different performance requirements to transactional data.
  - High throughput due to reading many rows from multiple tables
  - Long-running queries with aggregation and summarization
- Avoid locking your transactional workload by replicating business intelligence data to a dedicated analytics slave.
  - Use multiple slaves for different business units.
  - Configure indexes and storage engines on each slave to optimize for different analytics requirements.
    - Use replication filters or the BLACKHOLE storage engine to avoid replicating table data that you do not need to store on each slave.

Business intelligence and analytics: Business intelligence reports and analytical processing can be resource intensive and can take considerable time to execute. In a replicated environment, you can run such queries on a slave so that the master can continue to service the production workload without being affected by long-running and I/O-intensive reports.

Note

- Replication filters are described in the slide titled “Replication Filtering Rules.”
- Replicating with the BLACKHOLE storage engine is described in the slide titled “Replicating with the BLACKHOLE Storage Engine.”

Replication for Geographic Data Distribution

- Store geographically-relevant data on servers that are located near their primary users.
  - Use scaled-out replication at each location to increase the potential throughput for larger local user bases.
- Replicate some or all of this data to other locations.
  - Replicate location-independent data to all locations.
    - Users, products, categories, game records
  - Replicate summaries to central corporate servers.
    - Sales aggregates, campaign outcomes, game scores

Geographic data distribution: Companies with a distributed geographic presence can benefit from replication by having servers in each region that process local data and replicate that data across the organization. This provides the performance and administrative benefits of geographic proximity to customers and the workforce, while also enabling corporate awareness of data throughout the company.

Note: Multisource replication is ideal for geographically distributed servers. See the slide titled "Multisource Replication" later in this lesson.

Replicating with the BLACKHOLE Storage Engine

- The BLACKHOLE storage engine silently discards all data changes with no warnings.
  - It accepts DML requests to insert or update data, even though there cannot be any matching data.
  - It does not have any further intelligence or function.
- The binary log continues to record those changes successfully.
- Use BLACKHOLE for tables on a relay slave when it replicates all changes to further slaves, but does not itself need to store data in those tables.

You can use replication filters to prevent a server from replicating some data, but if you need to replicate that data to further slaves even though you are not storing it locally, use the BLACKHOLE storage engine to avoid saving that data locally.

For example, if you have a relay slave that is used solely for executing frequent long-running business intelligence reports on a small number of tables, you can configure all other replicated tables to use BLACKHOLE so that the server does not store data it does not need, while it replicates all changes to its slaves.

Note: Replication filters are described in the slide titled “Replication Filtering Rules.”

Replication for High Availability

Replication allows various high-availability use cases.

- **Controlled switchover**: Use a replica to act in place of a production server during hardware or system upgrades.
- **Server redundancy**: Perform a failover to a replica server in case of a system failure.
- **Online schema changes**: Perform a rolling upgrade in an environment with several servers to avoid an overall system outage.
- **Software upgrades**: Replicate across different versions of MySQL during an environment upgrade.
  - Slaves must run a later version than the master.
  - Queries issued during the upgrade must be supported by all versions used during the upgrade process.

Topics

- Replication overview
- Replication conflicts
- When to use replication
- Configuring replication

Configuring Replication

1. Draw a replication topology diagram.
2. Identify all servers that participate in replication.
   - Note that each replication master that replicates from another master is also a replication slave.
3. Configure a unique **`server-id`** for each server.
   - An unsigned 32-bit integer with a default value of 1.
   - Any master or slave server with a `server-id` of 0 refuses to replicate with other servers.
4. Configure each replication master.
5. Configure each replication slave to connect to its master.
6. Start replication on each replication slave with **`START SLAVE`**.

Configuring Replication Masters

- Enable TCP/IP networking.
  - Replication cannot use UNIX socket files.
- Enable the binary log.
  - During replication, each master sends its log contents to each slave.
- Create a user with the `REPLICATION SLAVE` privilege.

```
CREATE USER user@slave_hostname IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO user@slave_hostname;
```

  - Each slave must connect to a master to replicate from it.
  - On masters with multiple slaves, specify wildcards in the host name to match all slaves, or create multiple users.
- Back up the master database as a starting point for the slave.
- Record the log coordinates if you are not using GTIDs.
  - Use the `--master-data` option if you are using `mysqldump`.

Note

GTIDs refer to Global Transaction Identifiers. This lesson covers GTIDs in the slide titled “Global Transaction Identifiers (GTIDs).”

Configuring Replication Slaves

- Restore the backup from the master.
  - Verify that the `gtid_purged` variable is set if you are using GTIDs.
- Issue a **`CHANGE MASTER TO`** statement on each slave with the:
  - Network location of the master
    - `MASTER_HOST` and `MASTER_PORT` values
    - Optionally, use `MASTER_SSL` and related options to encrypt network traffic between masters and slaves during replication.
  - Replication account username and password (with the `REPLICATION SLAVE` privilege)
    - `MASTER_USER` and `MASTER_PASSWORD` values
  - Binary log coordinates from which to start replicating (if you are not using GTIDs)
    - The `MASTER_LOG_FILE` and `MASTER_LOG_POS` values store the binary log position from which the slave starts replicating.
    - Specify `MASTER_AUTO_POSITION=1` if you are using GTIDs.

Note

`MASTER_SSL` and related options are available only on SSL-enabled servers.

CHANGE MASTER TO

- Issue the `CHANGE MASTER TO…` statement on the slave to configure replication master connection details:

```
mysql> CHANGE MASTER TO
    ->    MASTER_HOST = 'host_name',
    ->    MASTER_PORT = port_num,
    ->    MASTER_USER = 'user_name',
    ->    MASTER_PASSWORD = 'password',
    ->    MASTER_LOG_FILE = 'master_log_name',
    ->    MASTER_LOG_POS = master_log_pos;
```

- Subsequent invocations of `CHANGE MASTER TO` retain the value of each unspecified option.
  - Changing the master's host or port also resets the log coordinates.
  - The following statement changes the password, but retains all other settings:

```
mysql> CHANGE MASTER TO MASTER_PASSWORD='newpass';
```

Finding Log Coordinates

- Execute `SHOW MASTER STATUS` on the master immediately after performing the backup.
  - Ensure that there is no activity on the master after the backup to guarantee that the log coordinates are those of the last event before the backup.
  - Example:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000014 |    51467 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (#.## sec)
```

- Use the `--master-data` option when you back up the master with `mysqldump` to find the log coordinates at the time of the backup.

Global Transaction Identifiers (GTIDs)

Global Transaction Identifiers (GTIDs) uniquely identify each transaction in a replication topology.

- Each GTID is of the form *<source-uuid>:<transaction-id>*. For example:

```
0ed18583-47fd-11e2-92f3-0019b944b7f7:338
```

- A GTID set contains a range of GTIDs:

```
0ed18583-47fd-11e2-92f3-0019b944b7f7:1-338
```

- Enable GTID mode with the following options:
  - **`gtid-mode=ON`**: Logs a unique GTID along with each transaction
  - **`enforce-gtid-consistency`**: Disallows events that cannot be logged in a transactionally safe way
  - **`log-slave-updates`**: Records replicated events to the slave's binary log

Note

`log-slave-updates` has the default value `ON` starting from MySQL server version 8.0.3.

Identifying the Source Server

The **`server_uuid`** value is:

- A universally unique identifier (UUID) stored in the **`auto.cnf`** file in the server's data directory
  - If the `auto.cnf` file does not exist, the server generates a new `auto.cnf` file containing a new `server_uuid` value.
  - You can inspect the `server_uuid` value by querying the system variable of the same name:

```
mysql> SELECT @@server_uuid\G
*************************** 1. row ***************************
@@server_uuid: 0ed18583-47fd-11e2-92f3-0019b944b7f7
1 row in set (#.## sec)
```

- Used in GTID to record which server the transaction has originally executed on
- Retained by all slaves in the replication chain so that the originating master for each transaction is identifiable

Logging Transactions

- When each server executes a transaction, it also records that transaction's ID in the **`gtid_executed`** variable.
  - Contains a set of GTIDs
  - Represents all transactions from local server, the immediate master, and any other upstream master servers

```
mysql> SELECT @@global.gtid_executed\G
*************************** 1. row ***************************
@@global.gtid_executed: bacc034d-4785-11e2-8fe9-0019b944b7f7:1-34,
c237b5cd-4785-11e2-8fe9-0019b944b7f7:1-9,
c9cec614-4785-11e2-8fea-0019b944b7f7:1-839
1 row in set (#.## sec)
```

- When the binary logs are purged, the GTID sets are removed from the binary log and stored in the **`gtid_purged`** system variable.
- Executing `RESET MASTER` causes both `gtid_executed` and `gtid_purged` to be set to an empty string.

Replication with GTIDs

Use **`CHANGE MASTER TO`**… to enable GTID replication:

- Tell the slave that transactions are identified by GTIDs:

```
CHANGE MASTER TO MASTER_AUTO_POSITION=1;
```

- You do not need to provide the log coordinates of the master (such as `MASTER_LOG_FILE` and `MASTER_LOG_POS`) because the slave sends the value of **`@@global.gtid_executed`** to the master. Therefore:
  - The master knows which transactions the slaves has executed
  - The master sends only those transactions that the slave has not already executed
- You cannot provide `MASTER_AUTO_POSITION` and log coordinates in the same `CHANGE MASTER TO…` statement.

Replication Filtering Rules

- Control the scope of replication by using ***replication filters***.
- Filters are server options that apply to a master or slave:
  - The master applies **`binlog-*`** filters when writing the binary log.
  - The slave applies **`replicate-*`** filters when reading the relay log.
- Use when different slaves in the environment serve different purposes.
  - Examples:
    - A server that is responsible for displaying web content does not require payroll or inventory data.
    - A server that provides sales data for management does not need data for web content or marketing data.

Applying Filtering Rules

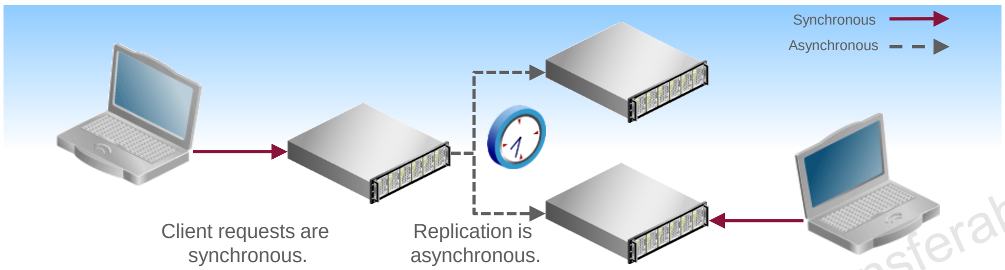
- Choose which events to replicate, based on:
  - Database:
    - `replicate-do-db, binlog-do-db`
    - `replicate-ignore-db, binlog-ignore-db`
  - Table:
    - `replicate-do-table, replicate-wild-do-table`
    - `replicate-ignore-table, replicate-wild-ignore-table`
- Consider precedence rules when applying filters:
  - Database filters apply before table filters.
  - Table wildcard filters `*-wild-*` apply *after* those that do not use wildcards.
  - The `*-do-*` filters apply *before* the respective `*-ignore-*` filters.

Be careful when using multiple filters. It is very easy to make mistakes, due to the complex order in which filters are applied. Because filters control what data is replicated, such mistakes are difficult to recover from. For this reason, do not mix different types of filters.
For example, if you use `replicate-wild-*`, do not use any non-wild `replicate-*` filters.

Temporary tables are not replicated when `binlog_format` is set to `ROW` or `MIXED`. To suppress replication of a specific temporary table when `binlog_format=STATEMENT`, use `replicate-ignore-table` or `replicate-wild-ignore-table`. Other replication filters have no effect on temporary tables.

Asynchronous Replication

- The slave requests the binary log and applies its contents.
  - The slave typically lags behind the master.
- The master does not care when the slave applies the log.
  - The master continues operating without waiting for the slave.



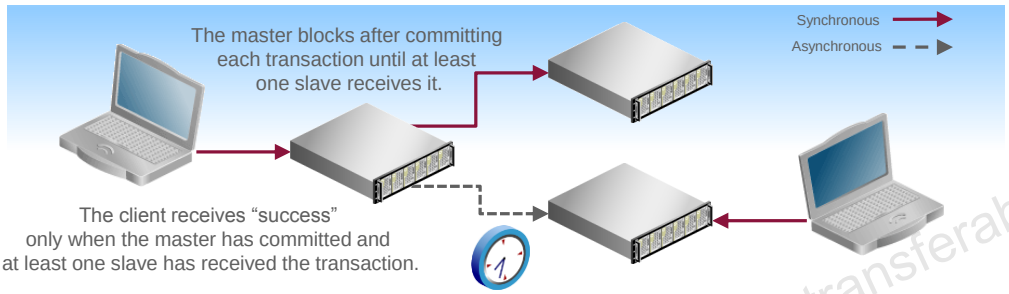
During MySQL replication in its default configuration, the master server accepts change events from clients, committing those events and writing them to the binary log. In a separate thread, the master streams the binary log to connected slaves. Because the master commits changes without waiting for a response from any slave, this is called *asynchronous* replication.

Most importantly, this means that slaves have not yet applied transactions at the time the master reports success to the application. Usually, this is not a problem. However, if the master crashes with data loss after committing a transaction but before the transaction replicates to any slave, the transaction is lost even though the application has reported success to the user.

If the master waited for all slaves to apply its changes before committing the transaction, replication would then be called *synchronous*. Although MySQL replication is not synchronous, MySQL NDB Cluster uses synchronous replication internally to ensure data consistency throughout the cluster, and MySQL client requests are synchronous because the client waits for a server response after issuing a query to the server.

Semisynchronous Replication

- Requires a plugin on the master and at least one slave
- Blocks each master event until at least one slave receives it
- Switches to asynchronous replication if a timeout occurs



During semisynchronous replication, the master blocks after committing a transaction until at least one semisynchronous slave acknowledges that it too has received the transaction. This means that at least one slave has received each transaction at the time the master reports success to the application. If the master crashes with data loss after committing a transaction and the application has reported success to the user, the transaction also exists on at least one slave.

Advantages and Disadvantages of Semisynchronous Replication

Semisynchronous replication:

- Ensures data integrity
- Results in reduced performance
  - The master waits for the slave to respond before committing the transaction.
    - The timeout period is controlled by the `rpl_semi_sync_master_timeout` variable value. The default is 10,000 ms (10 seconds).
    - If no response is received, the master still commits the transaction but reverts to asynchronous mode.
  - The extra time taken by each transaction includes:
    - The TCP/IP roundtrip to send the commit to the slave
    - The slave recording the commit in its relay log
    - The master waiting for the slave's acknowledgment of that commit
- Is suitable primarily for servers that are physically co-located, communicating over fast networks

Enabling Semisynchronous Replication

- Install the following plugins on the master and on at least one slave:
  - `rpl_semi_sync_master` on the master
  - `rpl_semi_sync_slave` on one or more slaves
- Enable the following options:
  - `rpl_semi_sync_master_enabled` on the master
  - `rpl_semi_sync_slave_enabled` on slaves
- Configure the `rpl_semi_master_timeout` variable if required.
  - Specifies the length of time a master will wait for a response from a semisynchronous slave.
    - After this amount of time, the master commits the transaction and reverts to aysnchronous mode.
  - The default value is 10,000 milliseconds (10 seconds).

Quiz

To use Global Transaction Identifiers in replication, you must use a `CHANGE MASTER TO…` statement to provide the binary log coordinates (file name and position) of the most recent transaction on the master that the slave has not yet applied.

a. True

b. False

Answer: b

Binary Logging

The binary log:

- Contains data and schema changes, and their time stamps
  - *Statement-based* or *row-based* logging
- Is used for point-in-time recovery from backup, full recovery from backup, and replication
- Rotates when:
  - MySQL restarts
  - It reaches its maximum size as set by `max_binlog_size`
  - You issue a `FLUSH LOGS` statement
- Can be inspected in various ways:
  - Metadata: `SHOW BINARY LOGS, SHOW MASTER STATUS`
  - Contents: `mysqlbinlog`

`FLUSH LOGS` in Replication Environments

By default, the MySQL server writes `FLUSH` commands to the binary log so that statements are replicated to the slaves. To prevent this from happening, use the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Binary Log Formats

- MySQL records information in the binary log in one of three different formats:
  - Row-based logging (the default method)
  - Statement-based logging
  - Mixed logging
- Change the binary log format by starting the global or session **`binlog_format`** server variable:

```
SET [GLOBAL|SESSION] BINLOG_FORMAT=[row|statement|mixed|default];
```

- You cannot set the `binlog_format` variable at run time:
  - From within a stored function or a trigger
  - If the NDB storage engine is enabled
  - If the session is currently using row-based replication and has open temporary tables

Row-Based Binary Logging

- Row-based binary logging:
  - Is the default binary logging format
  - Records the changes to individual table rows
    - Larger log is generated when many rows are updated.
    - The values recorded in the log are exactly the same as those sent to the storage engine.
    - A primary key is required on each table to correctly identify each row.
    - Fewer DML locks are required on slaves.
  - Always replays statements correctly, even if statements are non-deterministic
    - Example: `CURRENT_USER()` or `CONNECTION_ID()`
- Master and slave must use an identical table structure.

Statement-Based Binary Logging

- Contains the actual SQL statements
  - Can be used for auditing
  - Easier to identify statement for point-in-time recovery
- Includes both DDL (`CREATE, DROP,` and so on) and DML (`UPDATE, DELETE,` and so on) statements
- Saves disk space and network bandwidth due to the relatively small file sizes required
  - When one SQL statement modifies many rows
- Does not guarantee that non-deterministic statements replay correctly on a remote machine
  - If MySQL cannot make this guarantee, it issues a warning: Statement may not be safe to log in statement format

Mixed Format Binary Logging

- Uses statement-based logging by default
- Uses row-based logging when statements are non-deterministic:
  - Calls to `UUID()`, `USER()`, `FOUND_ROWS()`, `ROWS_COUNT()`, or any user-defined functions
  - When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked
  - When a statement refers to a system variable
  - Certain situations that involve views or temporary tables

Replication Logs

Slave servers maintain information about replication events.

- Relay log set:
  - Includes relay logs and relay log index file
  - Contains a copy of binary log events from the master
- Slave status logs:
  - Contain information needed to perform replication
    - Master connection details and log coordinates
    - Relay log coordinates
  - Are stored in tables (default) or files
    - `slave_master_info` and `slave_relay_log_info` tables in the `mysql` database
    - Default file names: `master.info` and `relay-log.info`

Relay logs: MySQL automatically manages the set of relay log files, removing files when it has replayed all events and creating a new file when the current file exceeds the maximum relay log file size (or `max_binlog_size` if `max_relay_log_size` is 0). The relay logs are stored in the same format as the binary logs; you can view them with `mysqlbinlog`. The slave maintains an index file to keep track of relay log files.

The relay log files are named *<host_name>*`-relay-bin.`*<nnnnnn>* by default, and the index file is named *<host_name>*`-relay-bin.index`. To make the server configuration immune to potential future host name changes, change these host name prefixes by setting the options `--relay-log` and `--relay-log-index`.

Slave status logs: The slave stores information about how to connect to the master, and the most recently replicated log coordinates of the master's binary log and the slave's relay log.

There are two such logs:

- **Master information:** This log contains information about the master server, including information such as the host name and port, credentials used to connect, and the most recently downloaded log coordinates of the master's binary log.
- **Relay log information:** This log contains the most recently executed coordinates of the relay log, and the number of seconds by which the slave's replicated events are behind those of the master.

Crash-Safe Replication

- Binary logging is crash-safe:
  - MySQL logs only complete events or transactions.
  - Use **`sync-binlog`** to improve safety.
    - Default value is 1; safest, the operating system writes to the file after every transaction.
    - Set the value to 0; best performance but highest possibility of transaction lost when server crashes as the operating system writes to the file according to its internal rules.
    - Set the value to any number greater than 1 to write after that number of transactions.
- Store slave status logs in tables for crash-safe replication.
  - Options: `master-info-repository` and `relay-log-info-repository`
    - Possible values are `TABLE` (the default) and `FILE`.
    - `TABLE` is crash-safe.

Crash-Safe Replication

As compared to previous versions of MySQL, MySQL 8.0 has set the default values to make the replication crash-safe.

The `sync-binlog` variable now defaults to 1 and both `master-info-repository` and `relay-log-info-repository` default to `TABLE`.

Also, the binary logging is enabled by default.

Multisource Replication

- Enables a replication slave to receive transactions from multiple masters simultaneously
  - It requires at least two masters and one slave.
  - The slave creates a *replication channel* for each master.
  - The slave must use `TABLE`-based repositories.
    - Multi-source replication is not compatible with `FILE`-based repositories.
- Does not attempt to detect or resolve conflicts
  - If this functionality is required, it is the responsibility of the application.
- Enables:
  - Backing up multiple servers to a single server
  - Consolidating data from multiple servers to a single server
  - Merging of table shards

Note

Multisource replication is compatible with auto-positioning.

Configuring Multisource Replication for a GTID-Based Master

1. Enable GTID-based transactions on the master by using `gtid_mode=ON`.
2. Create a replication user.
3. Verify that the slave is using `TABLE`-based replication repositories.
4. Execute `CHANGE MASTER TO` to add a new master to a channel by using a `FOR CHANNEL <channel>` clause.
   - Example: Add a new master with host name `master1` using port 3451 to channel `master-1`.

```
CHANGE MASTER TO MASTER_HOST='master1', MASTER_PORT=3451,
MASTER_USER='rpl', MASTER_PASSWORD='pass',
MASTER_AUTO_POSITION = 1
FOR CHANNEL 'master-1';
```

Configuring Multisource Replication for a Binary Log–Based Master

1. Enable binary logging on the master by using `--log-bin`.
2. Create a replication user.
3. Note the current binary log file and position.
   - `MASTER_LOG_FILE` and `MASTER_LOG_POSITION`
4. Verify that the slave is using `TABLE`-based replication repositories.
5. Execute `CHANGE MASTER TO` to add a new master to a channel by using a `FOR CHANNEL <channel>` clause.
   - Example: Add a new master with host name `master1` using port 3451 to channel `master-1`.

```
CHANGE MASTER TO MASTER_HOST='master1', MASTER_PORT=3451,
MASTER_USER='rpl', MASTER_PASSWORD='pass',
MASTER_LOG_FILE='master1-bin.000003', MASTER_LOG_POS=719,
FOR CHANNEL 'master-1';
```

Controlling Slaves in a Multisource Replication Topology

- If you have enabled multiple replication channels on a slave, you can execute `START SLAVE`, `STOP SLAVE`, or `RESET SLAVE` for specific channels with the `FOR CHANNEL` clause.
- If you do not use a `FOR CHANNEL` clause, the statement affects all currently configured replication channels.
- Examples:
  - Start replication on `channel-1`:

```
START SLAVE FOR CHANNEL 'channel-1';
```

  - Stop replication on `channel-1`:

```
STOP SLAVE FOR CHANNEL 'channel-1';
```

  - Reset all configured channels:

```
RESET SLAVE;
```

Summary

In this lesson, you should have learned how to:

- Describe MySQL replication
- Explain the role of replication in high availability and scalability
- Set up a MySQL replication environment
- Design advanced replication topologies

Practices

- 12-1: Quiz – Configuring Replication
- 12-2: Configuring Replication
- 12-3: Adding a New Slave
- 12-4: Enabling GTID and Configuring Circular Replication

Administering a Replication Topology

13

MySQL

ORACLE

Objectives

After completing this lesson, you should be able to:

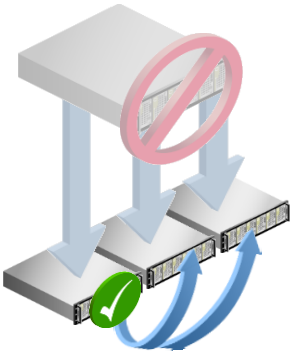
- Perform a controlled failover
- Explain MySQL replication threads
- Monitor MySQL replication
- Troubleshoot MySQL replication

Topics

- Failover
- Replication threads
- Monitoring replication
- Troubleshooting replication

Failover with Log Coordinates

1. In the binary logs, find the most recent event applied to each slave.
2. Select an up-to-date slave as a new master.
3. Identify the log coordinates on the new master to match the latest applied event on each slave.
4. Issue the correct `CHANGE MASTER TO…` on each slave.



Potential Problems when Executing a Failover with Log Coordinates

If the slaves are not all up-to-date when you fail over, you risk having an inconsistent replication topology:

- If the new master is behind a particular slave (that is, if the slave has already applied the events at the end of the new master's log), then the slave repeats those events.
- If the new master is ahead of a particular slave (that is, if the new master's binary log contains events that the slave has not yet applied), then the slave skips those events.

Avoiding Problems when Executing a Failover with Log Coordinates

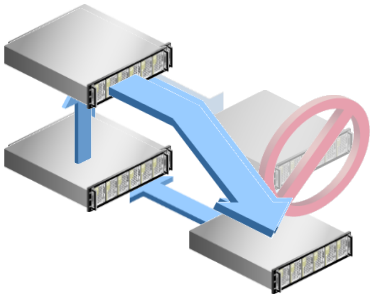
- Allow the SQL thread to apply all the events in the relay log.
- Select an up-to-date slave as the new master.
- Find the log coordinates on the new master that match the most recent event on each slave.
- If some slaves are more behind than others, the log coordinates that you provide in the `CHANGE MASTER TO…` statement are different from one slave to the next
  - You cannot simply issue a `SHOW MASTER STATUS` on the new master.
  - Instead, you must examine the binary logs to find the correct coordinates.
- In circular topologies with multiple masters accepting client updates, finding an up-to-date slave and identifying correct log coordinates can be very difficult.
  - Consider using Global Transaction Identifiers (GTIDs).

In circular topologies with multiple masters accepting client updates, finding an up-to-date slave and identifying correct log coordinates can be very difficult, because each slave applies operations in a different order to other slaves. To avoid this difficulty, use Global Transaction Identifiers (GTIDs).

Failover with GTIDs

When you use GTIDs, failover in a circular topology is trivial:

- On the slave of the failed master, bypass it by issuing a single `CHANGE MASTER TO` statement.
- Each server ignores or applies transactions replicated from other servers in the topology, depending on whether the transaction's GTID has already been seen or not.



Although GTIDs prevent the duplication of events that originated on a single server, they do not prevent conflicting operations that originated on different servers, as described in the slide titled "Complex Topologies" in the lesson titled "Configuring a Replication Topology." When reconnecting applications to your servers after a failover, you must be careful not to introduce such conflicts.

Topics

- Failover
- Replication threads
- Monitoring replication
- Troubleshooting replication

Replication Threads

When a slave connects to a master:

- The master creates a *Binlog dump thread*
  - Reads events from the binary log and sends them to the slave I/O thread
- The slave creates two threads by default:
  - *Slave I/O thread*
    - Reads events from the master's Binlog dump thread and writes them to the slave's relay log
  - *Slave SQL* (or "applier") *thread*
    - Applies relay log events on single-threaded slave
    - Distributes relay log events between worker threads on multithreaded slave

The Master's Binlog Dump Thread

- The master creates one instance of the Binlog dump thread for each connected slave.
- The Binlog dump thread:
  - Displays as `Binlog Dump GTID` if the slave uses auto-positioning
    - Configure auto-positioning with `CHANGE MASTER TO MASTER_AUTO_POSITION`.
  - Sends events within the binary log to the slave as those events arrive
    - For as long as the slave is connected

Single-Threaded Slaves

- Slaves are *single-threaded* by default.
  - Each slave uses a single SQL thread to process the relay log.
  - The benefit is that data within one database, when replicated by a single thread, is guaranteed to be consistent.
- Single-threaded slaves can result in *slave lag*, where the slave falls behind the master:
  - The master applies changes in parallel if it has multiple client connections, but it serializes all events in its binary log.
  - The slave executes these events sequentially in a single thread, which can become a bottleneck in high volume environments or when slave hardware is not powerful enough.

Multi-Threaded Slaves

- Use multi-threaded slaves to reduce slave lag.
- Set the `slave_parallel_workers` variable to a value greater than zero to create that number of worker threads.
  - With multiple worker threads, the slave SQL thread does not apply events directly, but delegates responsibility to worker threads.
- Set the `slave_parallel_type` variable to specify the parallelization policy.
  - `DATABASE` (default): Transactions that update different databases are applied in parallel.
  - `LOGICAL_CLOCK`: Transactions that are part of the same binary log group commit on a master are applied in parallel on a slave.
    - The master server can configure the binary log to record commit timestamps or write sets using the `binlog_transaction_dependency_tracking` variable.
    - Set `slave_preserve_commit_order` variable to `ON` to preserve the commit order.

`DATABASE` policy is only appropriate if data is partitioned into multiple databases which are being updated independently and concurrently on the master. There must be no cross-database constraints, as such constraints may be violated on the slave.

`LOGICAL_CLOCK` policy uses either the commit timestamps or write sets to determine which transactions can be applied in parallel. The dependency information is generated by the master server. Write sets can provide a higher level of parallelism compared to commit timestamps. With `slave_preserve_commit_order` enabled, the executing thread waits until all previous transactions are committed before committing. With this mode, a multithreaded slave never enters a state that the master was not in.

Controlling Slave Threads

- Start or stop both the I/O and SQL threads on a slave:

```
START SLAVE;
STOP SLAVE;
```

- Control threads individually:

```
START SLAVE IO_THREAD;
STOP SLAVE SQL_THREAD;
```

  - You cannot control individual worker threads in a multi-threaded slave, because these are controlled by the SQL thread.

- Start threads until a specified condition:

```
START SLAVE UNTIL SQL_AFTER_MTS_GAPS;
START SLAVE IO_THREAD UNTIL SQL_AFTER_GTIDS = '0ed18583-47fd-11e2-
92f3-0019b944b7f7:338';
```

- Disconnect the slave from the master:

```
RESET SLAVE [ALL]
```

Resetting the Slave

Disconnect the slave from the master for a clean start by issuing the **`RESET SLAVE`** command:

- Clears `master.info` and `relay.log` repositories
- Deletes all the relay logs
- Starts a new relay log file
- Resets any `MASTER_DELAY` specified in the `CHANGE MASTER TO` statement to zero
- Retains connection parameters so that you can restart the slave without executing `CHANGE MASTER TO`.
  - Issue **`RESET SLAVE ALL`** to reset the connection parameters.

Quiz

Which of the following statements are true? (Select all that apply.)

a. A single-threaded replication slave always has at least two threads.
b. The number of active worker threads in a multi-threaded slave might be affected by the number of databases in the replication network.
c. The slave I/O thread is responsible for processing the relay log.
d. The master sends binary log events to the slave using the Binlog dump thread.

Answer: a, b, d

Topics

- Failover
- Replication threads
- Monitoring replication
- Troubleshooting replication

Monitoring Replication

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State: Queueing master event to the relay log
                  ...
      Master_Log_File: mysql-bin.000005
  Read_Master_Log_Pos: 79
       Relay_Log_File: slave-relay-bin.000005
        Relay_Log_Pos: 548
Relay_Master_Log_File: mysql-bin.000004
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
                  ...
  Exec_Master_Log_Pos: 3769
                  ...
Seconds_Behind_Master: 8
```

Slave Thread Status

- The **`Slave_IO_Running`** and **`Slave_SQL_Running`** columns display the status of the slave's I/O and SQL thread, respectively.
- Possible values for each are:
  - `Yes`: The thread is currently running.
  - `No`: The thread is not running.
  - `Connecting`: The thread is running, but not yet connected to the master.

Master Log Coordinates

The **`Master_Log_File`** and **`Read_Master_Log_Pos`** columns identify the coordinates of the most recent event in the master's binary log that the I/O thread has transferred.

- Compare the master log coordinates with the coordinates shown when you execute `SHOW MASTER STATUS` on the master.
- If the values of `Master_Log_File` and `Read_Master_Log_Pos` are far behind those on the master, it indicates latency in the network transfer of events between the master and slave.

Relay Log Coordinates

- The `Relay_Log_File` and `Relay_Log_Pos` columns identify the coordinates of the most recent event in the slave's relay log that the SQL thread has executed.
- The `Relay_Master_Log_File` and `Exec_Master_Log_Pos` columns correspond to coordinates in the master's binary log .
  - If the columns' output is far behind the `Master_Log_File` and `Read_Master_Log_Pos` columns that represent the coordinates of the I/O thread:
    - There is latency in the SQL thread rather than the I/O thread
    - The log events are being copied faster than they are being executed
- The `Seconds_Behind_Master` column stores the number of seconds between the most recent relay log event and the current server time and helps to identify slave lag.

- **`Exec_Master_Log_Pos`:** On a multithreaded slave, `Exec_Master_Log_Pos` contains the position of the last point before any uncommitted transactions. This is not always the same as the most recent log position in the relay log, because a multithreaded slave might execute transactions on different databases in a different order from that represented in the binary log.
- **`Seconds_Behind_Master`:** This column provides the number of seconds between the time stamp (on the master) of the most recent event in the relay log executed by the SQL thread and the real time of the slave machine. A delay of this type normally occurs when the master processes a large volume of events in parallel that the slave must process serially, or when the slave's hardware is not capable of handling the same volume of events that the master can handle during periods of high traffic. If the slave is not connected to the master, this column is `NULL`.

  Note: This column does not show latency in the I/O thread or in the network transfer of events from the master.

Replication Slave I/O Thread States

The most commonly seen I/O thread states (in the **`State`** column of the `SHOW PROCESSLIST` output) are:

- **`Connecting to master`**
  - The thread is attempting to connect to the master.
- **`Waiting for master to send event`**
  - The slave thread has connected and is waiting for binary log events.
  - Can last a long time if the master is idle
  - Times out after `slave_read_timeout` seconds and attempts to reconnect
- **`Queuing master event to the relay log`**
  - The thread has read an event and is copying it to the relay log for processing by the SQL thread.

Replication Slave I/O Thread States

- **`Waiting to reconnect after a failed binlog dump request`**
  - The binary log dump request failed due to disconnection.
  - The thread enters this state while it sleeps and attempts to reconnect periodically.
  - Specify the interval between retries by using the `MASTER_CONNECT_RETRY` option of `CHANGE MASTER TO`.
- **`Reconnecting after a failed binlog dump request`**
  - The thread is trying to reconnect to the master.
- **`Waiting to reconnect after a failed master event read`**
  - A disconnect occurred during reading. The thread sleeps for a specified number of seconds before attempting to reconnect.
    - The default is 60 seconds.
  - Specify the number of seconds by using the `MASTER_CONNECT_RETRY` option of `CHANGE MASTER TO`.

Replication Slave I/O Thread States

- **`Reconnecting after a failed master event read`**
  - The thread is trying to reconnect to the master.
  - When the thread reconnects, the state becomes `Waiting for master to send event`.
- **`Waiting for the slave SQL thread to free enough relay log space`**
  - The combined size of the relay logs exceeds the value of `relay_log_space_limit`.
    - Only if non-zero. A zero value means there is no limit imposed on the size of the relay logs.
  - The I/O thread is waiting until the SQL thread frees enough space in the relay logs by processing and then deleting their contents.

Replication Slave SQL Thread States

The most commonly seen SQL thread states are:

- **`Waiting for the next event in relay log`**
  - The initial state before `Reading event from the relay log`.
- **`Reading event from the relay log`**
  - The thread has read an event from the relay log that can be processed.
- **`Making temp file (append) before replaying LOAD DATA INFILE`**
  - The thread is executing a `LOAD DATA` statement and is appending the data to a temporary file containing the data from which the slave will read rows.
- **`Slave has read all relay log; waiting for more updates`**
  - The thread has processed all events in the relay log files. It is now waiting for the I/O thread to write new events to the relay log.

Replication Slave SQL Thread States

- **`Waiting until … seconds after master executed event`**
  - The SQL thread has read an event but is waiting for the slave delay to lapse.
  - Set the delay with the `MASTER_DELAY` option of `CHANGE MASTER TO`.
- **`Waiting for an event from Coordinator`**
  - It appears only in worker threads on multi-threaded slaves.
  - A worker is waiting for the coordinating SQL thread to assign a job to the worker queue.

Monitoring Replication by Using Performance Schema

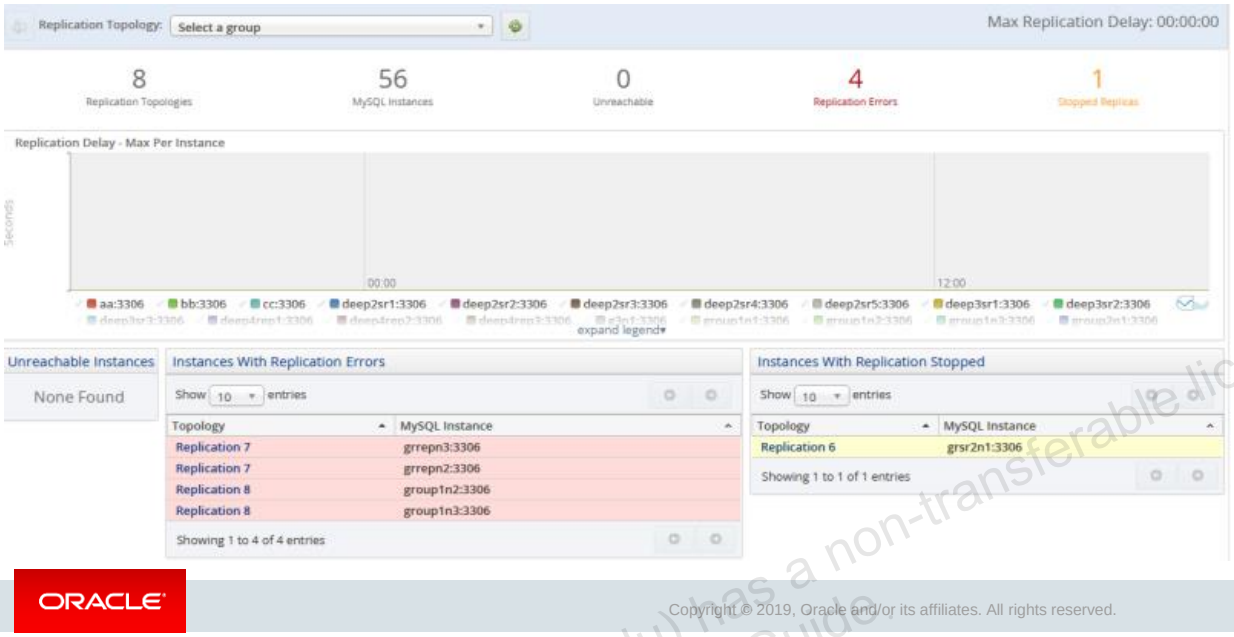
The Performance Schema contains `replication_*` tables that improve on the information available via `SHOW SLAVE STATUS`, enabling you to:

- Exercise more control over what information you want to display
- Retrieve better diagnostic information
  - `SHOW SLAVE STATUS` reports only the most recent coordinator and worker thread errors.
  - Retrieve details of the last transaction handled by each worker thread.
- Persist diagnostic information in tables or views
- Access replication metrics programmatically

Replication Tables in Performance Schema

Table Name	Contains
`replication_connection_configuration`	Configuration parameters for connecting to the master
`replication_connection_status`	Status of the current connection to the master
`replication_applier_configuration`	Configuration parameters for the transaction applier on the slave
`replication_applier_status`	Current status of the transaction applier on the slave
`replication_applier_status_by_coordinator`	Status of the SQL (or “applier”) thread in a multithreaded slave
`replication_applier_status_by_worker`	Status of applier thread in a single-threaded slave
`replication_applier_filters`	Information about the replication filters configured on specific replication channels
`replication_applier_global_filters`	Information about global replication filters (all channels)
`replication_group_members`	Network and status information for group members
`replication_group_member_stats`	Statistical information about group members and the transactions they participate in

MySQL Enterprise Monitor Replication Dashboard



The Replication Dashboard displays all information related to monitored replication groups. MySQL Enterprise Monitor supports monitoring of single-source tree hierarchy, circular replication, group replication, or complex, multi-level, multi-source hierarchies.

Navigate to the Replication page by choosing Replication under Dashboards. This page summarizes the state of your replication servers and enables you to drill down to see details about any source or replica. Using this page helps you avoid running the `SHOW SLAVE STATUS` command repeatedly on multiple servers; for consistency, the Replication page uses some of the same keywords as the output from `SHOW SLAVE STATUS`.

Topics

- Failover
- Replication threads
- Monitoring replication
- Troubleshooting replication

Troubleshooting MySQL Replication

- View the error log.
  - The error log can provide you with enough information to identify and correct problems in replication.
- Issue a `SHOW MASTER STATUS` statement on the master.
  - Logging is enabled if the position value is non-zero.
- Verify that both the master and slave have a unique non-zero server ID value.
  - The master and the slave must have different server IDs.
- Issue a `SHOW SLAVE STATUS` command on the slave.
  - `Slave_IO_Running` and `Slave_SQL_Running` display `Yes` when the slave is functioning correctly.
  - `Last_IO_Error` and `Last_SQL_Error` show the most recent error messages from the I/O and SQL threads.

Note: If you are unable to log in to the slave by using a valid account on the master, it might be because you created the user account with `CREATE USER` or `GRANT` on the master, and it replicated to the slave via the `mysql.user` table, but the replication process did not flush privileges. Use the `flush-privileges` command with `mysqladmin`, or execute `FLUSH PRIVILEGES` on the slave from another account.

Troubleshooting MySQL Replication

- Issue a `SHOW PROCESSLIST` command on the master and slave.
  - Review the state of the Binlog dump, I/O, and SQL threads.
- For a slave that suddenly stops working, check the most recently replicated statements.
  - The SQL thread stops if an operation fails due to a constraint problem or other error.
    - The error log contains events that cause the SQL thread to stop.
  - Review known replication limitations.
    - http://dev.mysql.com/doc/mysql/en/replication-features.html
  - Verify that the slave data has not been modified directly (outside of replication).

Examining the Error Log

- The error log stores information about when replication begins and about any errors that occur during replication.
- Always check the error log first.
- The following example shows a successful start and then a subsequent failure when replicating a user that already exists on the slave:

```
date-and-time 8449 [Note] Slave I/O thread: connected to master 'repl@127.0.0.1:3313',replication
started in log 'FIRST' at position 4
date-and-time 8449 [Note] Slave SQL thread initialized, starting replication in log 'mysql-
bin.000002' at position 108, relay log './slave-relay-bin.000003' position: 408
...
date-and-time 8449 [ERROR] Slave SQL: Error 'Operation CREATE USER failed for 'user'@'127.0.0.1''
on query. Default database: 'world_innodb'. Query: 'CREATE USER 'user'@'127.0.0.1' IDENTIFIED BY
PASSWORD '*2447D497B9A6A15F2776055CB2D1E9F86758182F'', Error_code: 1396
date-and-time 8449 [Warning] Slave: Operation CREATE USER failed for 'user'@'127.0.0.1' Error_code:
1396
date-and-time 8449 [ERROR] Error running query, slave SQL thread aborted. Fix the problem, and
restart the slave SQL thread with "SLAVE START". We stopped at log 'mysql-bin.000002' position 460
```

Examining the Error Log

The following sequence shows a failure of the I/O thread reading from the master's binary log:

```
date-and-time 8823 [Note] Slave I/O thread: connected to master
'repl@127.0.0.1:3313',replication started in log 'mysql-bin.000002' at position 1172
date-and-time 8823 [ERROR] Read invalid event from master: 'Found invalid event in
binary log', master could be corrupt but a more likely cause of this is a bug
date-and-time [ERROR] Slave I/O: Relay log write failure: could not queue event from
master, Error_code: 1595
date-and-time 8823 [Note] Slave I/O thread exiting, read up to log 'mysql-bin.000003',
position 4
```

SHOW SLAVE STATUS Error Details

- **Last_IO_Error, Last_SQL_Error**:
  - The error message of the most recent error that caused, respectively, the I/O thread or SQL thread to stop.
  - During normal replication, these fields are empty.
  - If an error occurs and causes a message to appear in either of these fields, the error message also appear in the error log.
- **Last_IO_Errno, Last_SQL_Errno**:
  - The error number associated with the most recent error that caused, respectively, the I/O or SQL thread to stop.
  - During normal replication, these fields contain the number 0.
- **Last_IO_Error_Timestamp, Last_SQL_Error_Timestamp**:
  - The time stamp of the most recent error that caused, respectively, the I/O thread or SQL thread to stop.
  - During normal replication, these fields are empty.

Checking I/O Thread States

If the slave is running, issue `SHOW PROCESSLIST` to check the I/O thread states connecting to the master:

- Verify privileges for the user being used for replication on the master.
- Verify that the host name and port are correct for the master.
- Verify that networking has not been disabled on the master or the slave (with the `--skip-networking` option).
- Attempt to ping the master to verify that the slave can reach the master.

Monitoring Multisource Replication

- Execute **SHOW SLAVE STATUS FOR CHANNEL *<channel name>***.
- Alternatively, query the Performance Schema `replication_*` tables, specifying the channel name.
  - The `CHANNEL_NAME` field exists in every Performance Schema replication table.

```
mysql> SELECT * FROM replication_connection_status
    -> WHERE CHANNEL_NAME='master1'\G;
*************************** 1. row ***************************
CHANNEL_NAME: master1
GROUP_NAME:
SOURCE_UUID: 046e41f8-a223-11e4-a975-0811960cc264
THREAD_ID: 24
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 046e41f8-a223-11e4-a975-0811960cc264:4-37
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
```

ORACLE

The example in the slide demonstrates querying the Performance Schema `replication_connection_status` table to monitor the status of the I/O thread that handles the slave server connection to the master server on replication channel `master1`.

Summary

In this lesson, you should have learned how to:

- Perform a controlled failover
- Explain MySQL replication threads
- Monitor MySQL replication
- Troubleshoot MySQL replication

Practices

- 13-1: Monitoring Replication with MySQL Enterprise Monitor
- 13-2: Troubleshooting Replication Errors
- 13-3: Performing a Replication Failover

Achieving High Availability with MySQL InnoDB Cluster

14

Objectives

After completing this lesson, you should be able to:

- Describe MySQL InnoDB Cluster and Group Replication
- List typical use cases for MySQL InnoDB Cluster
- Contrast the two different modes for deployment
- Configure a MySQL InnoDB Cluster
- Administer a MySQL InnoDB Cluster

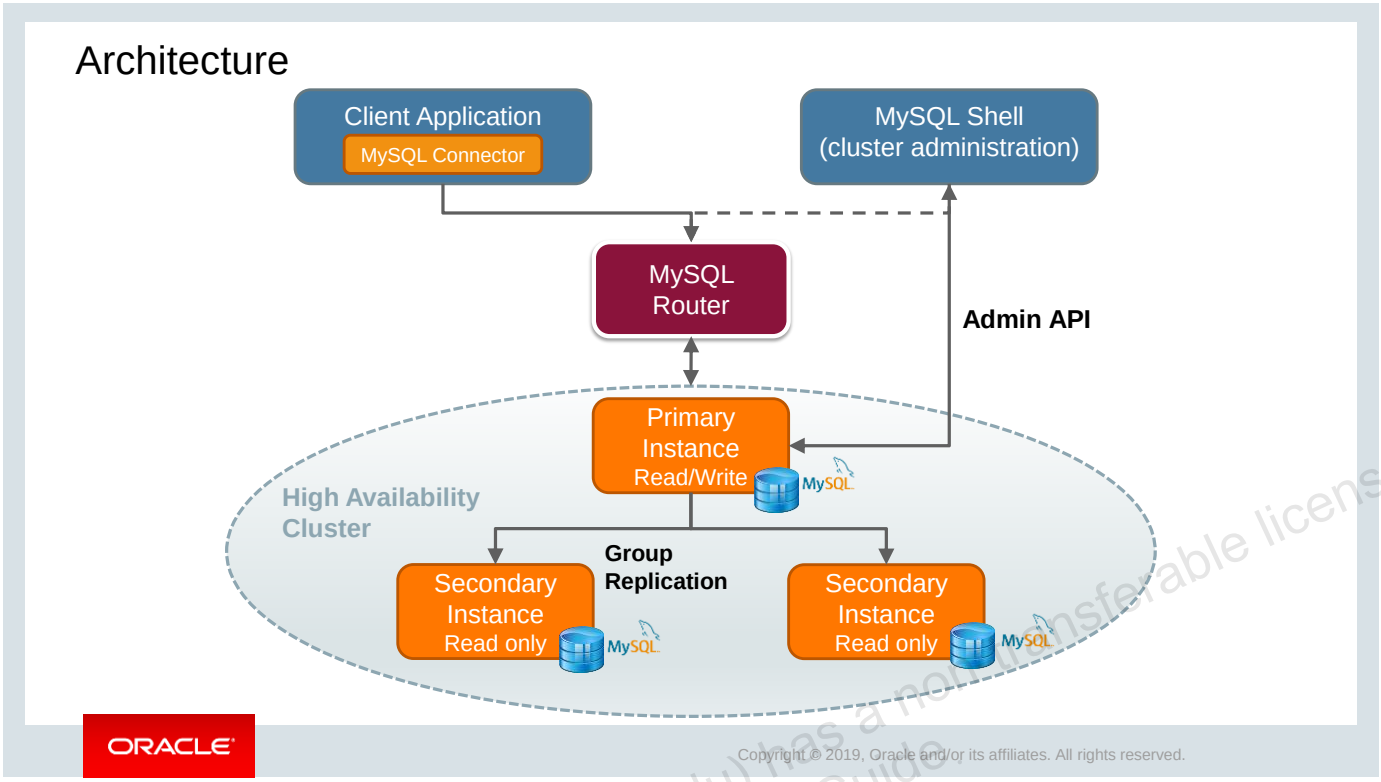
Topics

- Overview and architecture
- Tools
- Configuring a cluster
- Administering a cluster

What Is MySQL InnoDB Cluster?

- Provides a complete and scalable high availability solution for MySQL
  - Easy to configure and administer a group of server instances in a cluster
  - Requires at least three servers for the group to achieve consensus
- Uses **MySQL Group Replication** to replicate data between all servers in the group
  - The **AdminAPI** removes the need to work with group replication directly.
  - You work with the AdminAPI via **MySQL Shell** using your choice of Python, JavaScript, or SQL commands.
- Manages failover automatically
  - If a server in the group goes down, the cluster reconfigures itself.
- Enables clients to connect to the group transparently
  - Clients connect to the group via **MySQL Router** and do not have to know the details of the individual instances within the group.

To use MySQL InnoDB Cluster, you must download and install the MySQL Shell and MySQL Router software.



The Cluster relies on MySQL Group Replication, which is installed on each server instance in the cluster. Group Replication is a MySQL Server plugin that enables you to create elastic replication topologies that can reconfigure themselves automatically if a server in the cluster goes offline. There must be at least three servers to form a group that can provide high availability. Groups can operate in a single-primary mode where only one server accepts updates at a time, or multi-primary mode, where all servers can accept updates, even if they are issued concurrently.

The MySQL Group Replication plugin was introduced in 5.7, but it is tricky to work with directly. MySQL InnoDB Cluster introduces new components that are tightly integrated and make Group Replication much easier to set up and administer. These include MySQL Router, which sits between the application and the cluster and routes the traffic to the appropriate cluster instance. If the primary instance goes down, the cluster automatically promotes another instance to take its place, and the MySQL Router starts routing traffic to the new primary instance. All this happens without DBA intervention.

You administer the cluster by using MySQL Shell. MySQL Shell is a new interactive interface that enables you to administer MySQL via the new Admin API, using familiar JavaScript, Python, or SQL commands.

MySQL Group Replication Plugin

Group Replication is a plugin for MySQL that enables a group of servers to replicate data between them, and provides:

- Automatic handling of server failover
- Automatic reconfiguration of groups when members join or leave the group due to crashes, failures, or reconnects
- Fault tolerance
- Conflict resolution
- A highly available, replicated database

How Group Replication Works

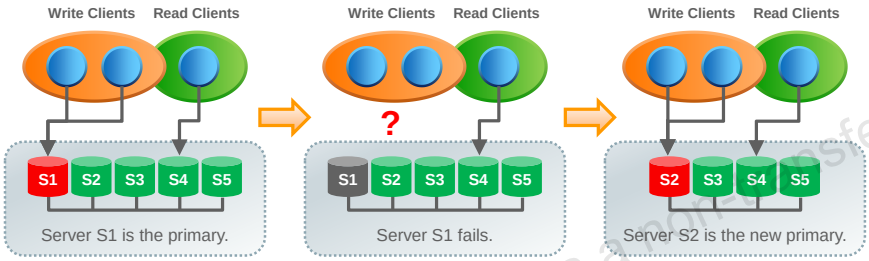
- Servers belong to a *replication group.*
  - A replication group can contain up to nine servers.
    - At least three servers are required to achieve consensus
  - Group Replication uses global transaction identifiers (GTIDs).
  - The group is defined by a UUID
- Group membership is managed automatically.
  - A server that joins or rejoins the group will automatically synchronize with the others.
  - Servers can leave and join the group at any time.
- Replication groups operate in one of two modes:
  - **Single-primary**: One server in the group accepts updates.
  - **Multi-primary**: All servers in the group accept updates.
- Changes are replicated to all members of the group.

Member servers are configured to belong to a *replication group*. Clients write to one or more members, depending on the Group Replication mode. Changes are replicated to all members of the replication group.

Groups can operate in *single-primary mode* with automatic primary election, where only one member accepts updates. Alternatively, groups can be deployed in *multi-primary mode*, where all members accept updates, even if they are issued concurrently.

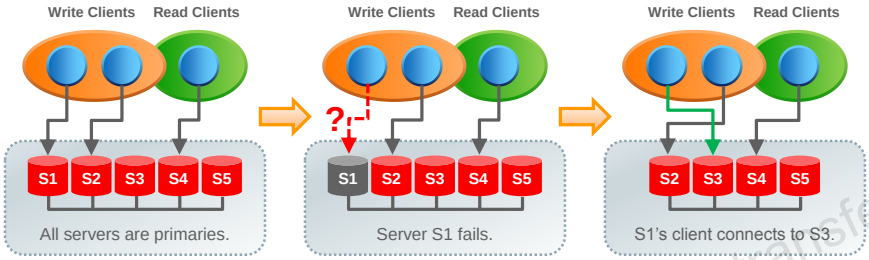
Single-Primary Mode

- One group member accepts writes (the *primary*).
- The remaining group members act as read-only hot standbys (*secondaries*).
- This is the default mode.
  - Applications and developers interact only with a single server for updates.
  - This avoids some of the limitations of multi-primary mode.
  - Applications can connect to any secondary nodes to read data.
- If the primary fails, the group automatically elects a new primary:



Multi-Primary Mode

- All servers can receive updates, even when executed concurrently.
- If one server fails, clients can connect to any other server in the replication group.



Conflict Resolution

- Each server in the group executes transactions independently, but all servers must agree on the decision.
  - Read-write transactions commit when they are received by the group.
    - For the transaction to be applied, the majority of members must agree (“certify”) the order of the transaction within the global sequence of transactions.
  - Read-only transactions do not require group approval and commit immediately.
- If two concurrent transactions affect a single row, there is a conflict.
  - The first transaction that commits “wins.” The others roll back.
- If a network partition results in a “split brain” situation where group members are unable to agree, then the system does not achieve consensus and requires manual intervention.

The “first transaction to commit wins” rule also applies to single-primary mode.

Use Cases

- Elastic replication:
  - Environments where the number of servers involved in the replication infrastructure is very fluid.
- Highly available shards:
  - Sharding is a popular approach to write scale-out.
  - Each shard can map to a replication group.
- As an alternative to standard master-slave replication:
  - Using single-primary mode, the following operations are automatic:
    - Assignment of primary/secondary roles
    - Election of new primary when current primary fails
    - Setup of read/write modes on primaries and secondaries
    - Global consistent view of which server is the primary

Group Replication: Requirements and Limitations

Required

- For communication:
  - Local network with servers in close proximity to reduce latency
- For conflict detection:
  - InnoDB storage engine
  - Primary key on every table
  - GTIDs are enabled
- For replication:
  - Binary logging and slave updates enabled
  - Binary log `ROW` format
  - Metadata must be stored in `TABLE` format
  - `transaction_write_set_extraction=XXHASH64`

Forbidden

- General:
  - Binary log events checksum
  - Table locks and named locks
  - Replication filters
- In multi-primary mode:
  - `SERIALIZABLE` isolation level
  - Cascading foreign keys
  - Concurrent DDL and DML statements that access the same object on different members

ORACLE

You must ensure that `--transaction_write_set_extraction=XXHASH64` (the default option), so that while collecting rows to log them to the binary log, the server collects the write set as well. The write set is based on the primary keys of each row and is a simplified and compact view of a tag that uniquely identifies the row that was changed. Group replication uses this tag to detect conflicts.

When running in multi-primary mode, you cannot execute data definition language (DDL) and data manipulation language (DML) statements on the same object on different members, because it may result in undetectable conflicts.

Quiz

Which of the following statements are correct about Group Replication in single-primary mode? (Select all that apply.)

a. All servers accept writes to improve scalability.
b. One server accepts writes, the others are read-only.
c. There must be at least two instances, one for writes and one for reads.
d. A "split brain" situation always results when the secondaries lose connection to the primary.

Answer: b

Topics

- Overview and architecture
- Tools
- Configuring a cluster
- Administering a cluster

MySQL Shell (`mysqlsh`)

- Is an advanced client and code editor for MySQL
- Provides scripting capabilities by using JavaScript, Python, or SQL commands
  - You can enter commands interactively, or execute them in batches.
  - You can switch between languages by entering `\js`, `\py`, or `\sql`, respectively.
- Enables access to MySQL features via APIs
  - **XDevAPI**: Communicate with a MySQL Server running the X Plugin to work with both relational and document data in the MySQL Document Store.
  - **AdminAPI:** Configure and administer a MySQL InnoDB Cluster.
    - Requires Python 2.7 or later to be installed, regardless of the choice of scripting language
- Supports output in tab delimited, table, and JSON (JavaScript Object Notation) formats
- Interacts with a MySQL server via a global Session object
  - If using Python and JavaScript, you create the Session object by calling the `getSession()` method on the `mysqlx` module.
  - If using SQL, the Session object is created when the client connects.

The examples in this course use JavaScript (the default scripting language for MySQL Shell).

Using MySQL Shell to Execute a Script

Examples:

- Loading JavaScript code from a file for batch processing:

```
$ mysqlsh  --file script.js
```

- Redirecting a JavaScript file to standard input for execution:

```
$ mysqlsh < script.js
```

- Redirecting SQL to standard input for execution:

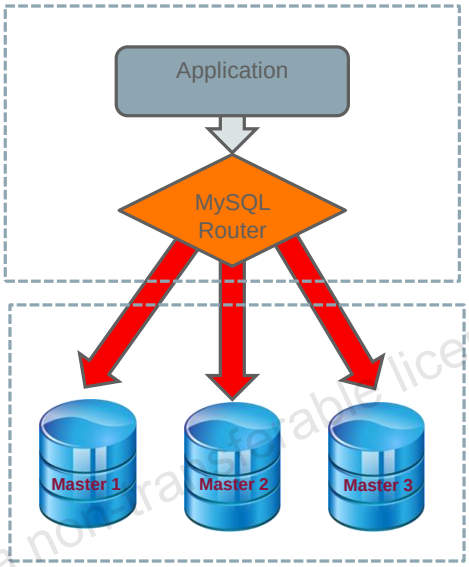
```
$ echo "show databases;" | mysqlsh --sql --uri root@127.0.0.1:3306
```

- Making a batch file executable (Linux only):

```
#!/usr/local/mysql-shell/bin/mysqlsh --file
print("Hello World\n");
```

MySQL Router (`mysqlrouter`)

- Acts as middleware that provides transparent routing between client applications and back-end MySQL servers
- Manages failover by automatically routing connections
  - No custom code required; uses a load balancing policy
- Provides load balancing
  - Distributes database connections across a pool of servers for performance and scalability
- Implements a pluggable architecture
  - Developers can extend the product and create plugins for custom use cases.



Topics

- Overview and architecture
- Tools
- Configuring a cluster
- Administering a cluster

Deployment Scenarios

You can configure a MySQL InnoDB Cluster using one of two methods:

- **Sandbox deployment:** Enables you to test MySQL InnoDB Cluster locally, prior to deployment on production servers
- **Production deployment:** Enables you to deploy the individual instances that make up a cluster on multiple host machines in a network

Deploying Sandbox Instances and Creating the Cluster

1. Deploy three sandbox instances on ports 3310, 3320, and 3330:

```
mysql-js> dba.deploySandboxInstance(3310)
mysql-js> dba.deploySandboxInstance(3320)
mysql-js> dba.deploySandboxInstance(3330)
```

2. Connect to the seed instance (where the data resides) and create the cluster:

```
mysql-js> \connect root@localhost:3310
mysql-js> var cluster = dba.createCluster('mycluster')
```

3. Add instances:

```
mysql-js> cluster.addInstance(3320)
mysql-js> cluster.addInstance(3330)
```

The `dba` shell global variable represents the AdminAPI. You call its `deploySandboxInstances` method, passing the required TCP port for the instance as a parameter.

You then connect to the instance by using the `shell.connect('user@localhost:port')` method, or its shorthand equivalent `\connect user@server:port`, as demonstrated in the slide example. This example uses a URI string for the connection details, but you can also use a dictionary.

When you create a cluster using `dba.createCluster()`, the operation returns a `Cluster` object which you can assign to a variable. You use this object to work with the cluster, for example, to add instances or check the cluster's status. If you want to retrieve a cluster again at a later date, for example, after restarting MySQL Shell, use `dba.getCluster(`*`clustername, options`*`)`. Set the `connectToPrimary` option to query for and connect to a primary instance. For example:

```
cluster1 = dba.getCluster(mycluster, {connectToPrimary:false})
```

MySQL InnoDB Cluster relies on `SET PERSIST` to persist instance configuration settings automatically. This is only available on instances running version 8.0.11 of MySQL Server or later with `persisted_globals_load=ON` (the default setting). On local instances that support automatic persistence, the configuration settings are stored in the instance's `mysqld-auto.cnf` file. If a local instance does not support automatic persistence, the AdminAPI attempts to write these changes to the instance's option file.

Production Deployment

1. Perform the following steps for each machine that will be part of the cluster.
   - Ensure that the machine's name is resolvable by other machines in the cluster by one of the following methods:
     - Map the IP addresses of all the other machines to a host name.
     - Set up a DNS service.
     - Configure the `--report_host` variable in the instance MySQL configuration file.
2. Verify that each instance is correctly configured for MySQL InnoDB Cluster by executing `dba.checkInstanceConfiguration(`*`connection details`*`)`.
   - The output of the method call lists any required changes. You can accept these changes by executing `dba.configureInstance(`*`connection details`*`)`.
3. Create the Cluster by executing `dba.createCluster(`*`cluster name`*`)`.
4. Add instances by executing `dba.addInstance(`*`connection details`*`)`.

If a remote instance is not running version 8.0.11 of MySQL Server or later with `persisted_globals_load=ON` (the default setting), the AdminAPI commands cannot automatically write to the instance's option file to persist any configuration options. It can read configuration options from the file, but it cannot change them. If you need to make changes to the remote instance's option file to configure it for MySQL InnoDB Cluster usage, you must connect to the server using a tool such as SSH, and run MySQL Shell directly on the instance to configure it locally by using `dba.configureLocalInstance()`.

Connecting Clients to the Cluster

1. Install MySQL Router on the same host as the application.
2. Bootstrap MySQL Router with the metadata server on the seed instance:

```
$ mysqlrouter --bootstrap user@hostname:port --directory=directory_path
```

   - This creates read/write and read-only TCP ports.

3. Start MySQL Router:

```
$ directory_path/start.sh
```

4. Connect the client to the appropriate TCP port.
   - For example, using MySQL Shell to connect to the read/write port 6446 of mysqlrouter as the `root` user

```
$ ./bin/mysqlsh --uri root@localhost:6446
```

5. To verify which machine you are connected to, you can query the `port` and `hostname` server variables.

The `--directory` option configures MySQL Router to run from a self-contained directory. This enables you to deploy multiple instances of the router in the same host without root privileges.

Topics

- Overview and architecture
- Tools
- Configuring a cluster
- Administering a cluster

Managing Sandbox Instances

Manage sandbox instances with the following functions:

- **`dba.deploySandboxInstance(port)`**: Create a new instance.
- **`dba.startSandboxInstance(port)`**: Start a sandbox instance.
- **`dba.stopSandboxInstance(port)`**: Stop a running instance gracefully, unlike `dba.killSandboxInstance()`.
- **`dba.killSandboxInstance(port)`**: Stop a running instance immediately. Useful for simulating unexpected halts.
- **`dba.deleteSandboxInstance(port)`**: Remove a sandbox instance from your file system.

Note

Each of these functions also accepts an optional options object as a second parameter, that affects the result of the operation.

Checking the Status of a Cluster

```
mysql-js> var cluster = dba.getCluster("mycluster")
mysql-js> cluster.status()
{
    "clusterName": "mycluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "localhost:3320",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "localhost:3310": {
                "address": "localhost:3310",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "localhost:3320": {
                  ...
            },
            "localhost:3330": {
                ...
             }
                ...
```


Note

The way in which you connect to the cluster affects the output of `cluster.status()`. If you connect via a read-only connection, you will only be able to see status information that relates to the secondary instances. To see status information for the primary instance(s), use a read/write connection.

Viewing the Structure of a Cluster

```
mysql-js> cluster.describe();
{
    "clusterName": "mycluster",
    "adminType": "local",
    "defaultReplicaSet": {
        "name": "default",
        "instances": [
            {
                "name": "localhost:3310",
                "host": "localhost:3310",
                "role": "HA"
            },
            {
                "name": "localhost:3320",
                "host": "localhost:3320",
                "role": "HA"
            },
            {
                "name": "localhost:3330",
                "host": "localhost:3330",
                "role": "HA"
            }
        ]
    }
}
```


The output of the `cluster.describe()` function shows the structure of the InnoDB cluster including all of its configuration information.

Rescanning a Cluster

- If an instance's configuration is changed by any means other than via the AdminAPI, you must rescan the cluster to update the InnoDB Cluster metadata.
  - For example, you manually added a new instance to the Group Replication group.
- Execute **`cluster.rescan()`**
  - Displays any discovered instances and asks you if you want to add them to your cluster

Removing Instances from the Cluster

Remove an instance from the cluster by calling the **`cluster.removeInstance()`** method:

```
mysql-js> cluster.removeInstance('user@hostname:port')
```

Customizing a MySQL InnoDB Cluster

- **Replication group name**: Execute `dba.createCluster(`*`cluster name`*`, { groupName: `*`new name`*`} )`
  - *`new name`* must be a valid UUID.
  - This also sets the `group_replication_group_name` variable.
- **Local address**: Execute `dba.createCluster(`*`cluster name`*`, { localAddress: `*`host:port`*`)})` and `cluster.addInstance(`*`instance`*`, { localAddress: `*`host`*`:`*`port`*`)})`.
  - The address must be accessible to all cluster instances and reserved for internal cluster communication only.
  - This also sets the `group_replication_local_address` system variable.
- **Seed instances**: Execute `dba.CreateCluster(instance, {groupSeeds: `*`seed list`*`) and cluster.addInstance(instance, {groupSeeds: `*`seed list`*`)}`
  - Seed list is a comma-separated list of seed instances specified as follows: `"`*`host1`*`:`*`port1`*`", "`*`host2`*`:`*`port2`*`", ...`

AdminAPI automatically configures the cluster group name, local address, and seed instances, which is recommended for most deployments. Advanced users can override these defaults by executing the methods shown in the slide.

- **Replication group name:** The name of the group the cluster instances belong to
- **Local address:** The address an instance provides for connections from other instances
- **Seed instances:** The instances that are contacted when a new instance joins the cluster

Checking the State of an Instance

- Execute **`cluster.checkInstanceState(`*`instance`*`)`** to verify the instance GTID state in relation to the cluster.
  - Analyzes the instance executed GTIDs with the executed/purged GTIDs on the cluster to determine if the instance is valid for the cluster.
- The output of this method is one of the following:
  - **`OK new`**: The instance has not executed any GTID transactions; therefore, it cannot conflict with the GTIDs executed by the cluster.
  - **`OK recoverable`**: The instance has executed GTIDs, which do not conflict with the executed GTIDs of the cluster seed instances.
  - **`ERROR diverged`**: The instance has executed GTIDs, which diverge with the executed GTIDs of the cluster seed instances.
  - **`ERROR lost_transactions`**: The instance has more executed GTIDs than the executed GTIDs of the cluster seed instances.

Rejoining a Cluster

If an instance that has left the cluster (for example, due to a lost connection) does not rejoin it automatically, execute `cluster.rejoinInstance(`*`instance`*`)`.

This might be because the instance was unable to persist its configuration information. If this is the case:

1. Execute `cluster.rejoinInstance()`.
2. Connect to the instance via SSH or a similar remote access tool.
3. Run MySQL Shell locally on the instance and execute `dba.configureLocalInstance()` to persist its configuration.

Restoring Quorum Loss

If an instance fails, then the cluster can lose its quorum, which is the ability to elect a new primary. Re-establish quorum with **`cluster.forceQuorumUsingPartitionOf()`**.

```
mysql-js> var cluster = dba.getCluster("mycluster")

  // The cluster lost its quorum and its status shows "status": "NO_QUORUM"

mysql-js> cluster.forceQuorumUsingPartitionOf("localhost:3310")

  Restoring replicaset 'default' from loss of quorum, by using the partition composed of
      [localhost:3310]

  Please provide the password for 'root@localhost:3310': ******
  Restoring the InnoDB cluster ...

  The InnoDB cluster was successfully restored using the partition from the instance
      'root@localhost:3310'.

  WARNING: To avoid a split-brain scenario, ensure that all other members of the replicaset
  are removed or joined back to the group that was restored.
```

Recovering the Cluster from a Major Outage

If a cluster comes to a complete standstill, perform the following steps:

1. Restart the cluster instances.
   - For example, in a sandbox deployment:

```
mysql-js> dba.startSandboxInstance(3310)
mysql-js> dba.startSandboxInstance(3320)
mysql-js> dba.startSandboxInstance(3330)
```

2. Connect to one instance and run MySQL Shell.
3. From MySQL Shell, connect to a cluster and execute **`dba.rebootClusterFromCompleteOutage()`**:

```
mysql-js> \connect 'root@localhost:3310'
mysql-js> var cluster = dba.rebootClusterFromCompleteOutage()
```

Enabling and Disabling Writes with `super_read_only`

Whenever Group Replication stops, the **`super_read_only`** variable is set to **`ON`** to ensure no writes are made to the instance. When you try to use such an instance with the following AdminAPI commands, you are given the choice to set `super_read_only=OFF` on the instance:

- `dba.configureInstance()`
- `dba.configureLocalInstance()`
- `dba.createCluster()`
- `dba.rebootClusterFromCompleteOutage()`
- `dba.dropMetadataSchema()`

Quiz

How do you persist a remote instance's configuration settings for MySQL InnoDB Cluster when the version of MySQL Server is prior to 8.0.11?

a. Manually configure the remote instance's `/etc/my.cnf` file.
b. Pass `persist='true'` as an option to `cluster.createInstance()`.
c. Ensure that the remote instance has `persisted_globals_load` set to `ON`.
d. Connect to the remote instance, and use MySQL Shell on the instance to execute `dba.configureLocalInstance()`.

Answer: d

Securing a Cluster

If the server instance provides SSL encryption, then it is automatically enabled on the seed instance when you execute `dba.createCluster()`. Use the **`memberSslMode`** option in the call to `dba.getCluster()` to change the SSL mode:

- **`dba.createCluster({memberSslMode:'DISABLED'})`**: SSL encryption is disabled for the seed instance in the cluster.
- **`dba.createCluster({memberSslMode:'REQUIRED'})`**: SSL encryption is enabled for the seed instance in the cluster. If it cannot be enabled, an error is raised.
- **`dba.createCluster({memberSslMode:'AUTO'})`** (the default): SSL encryption is automatically enabled if the server instance supports it, or disabled if the server does not support it.

Securing Instances

When you issue the `cluster.addInstance()` and `cluster.rejoinInstance()` commands, SSL encryption on the instance is enabled or disabled based on the setting found for the seed instance. Use the `memberSslMode` in the call to `cluster.addInstance()` or `cluster.rejoinInstance()` for more control:

- **`memberSslMode:'DISABLED'`**: Ensures that SSL encryption is disabled for the instance in the cluster
- **`memberSslMode:'REQUIRED'`**: Forces SSL encryption to be enabled for the instance in the cluster
- **`memberSslMode:'AUTO'`** : (Default) Automatically enables or disables encryption based on the setting used by the seed instance (other members of the cluster) and the available SSL support provided by the instance itself

Note

For sandbox instances, `cluster.deploySandboxInstance()` tries to deploy instances with SSL encryption by default where possible.

Creating a Server Whitelist

- The cluster maintains a list of approved servers that belong to the cluster, known as a whitelist.
  - Only servers in the whitelist can join the cluster.
  - The default whitelist is the private network addresses that the server has network interfaces on.
- Create your own whitelist with the **`ipWhiteList`** option to `dba.createCluster()`, `cluster.addInstance()`, or `cluster.rejoinInstance()` to improve security.
  - Pass the servers as a comma-separated list, surrounded by quotes.
  - Configure the `group_replication_ip_whitelist` system variable on the instance.

Dissolving a Cluster

To completely dissolve a cluster:

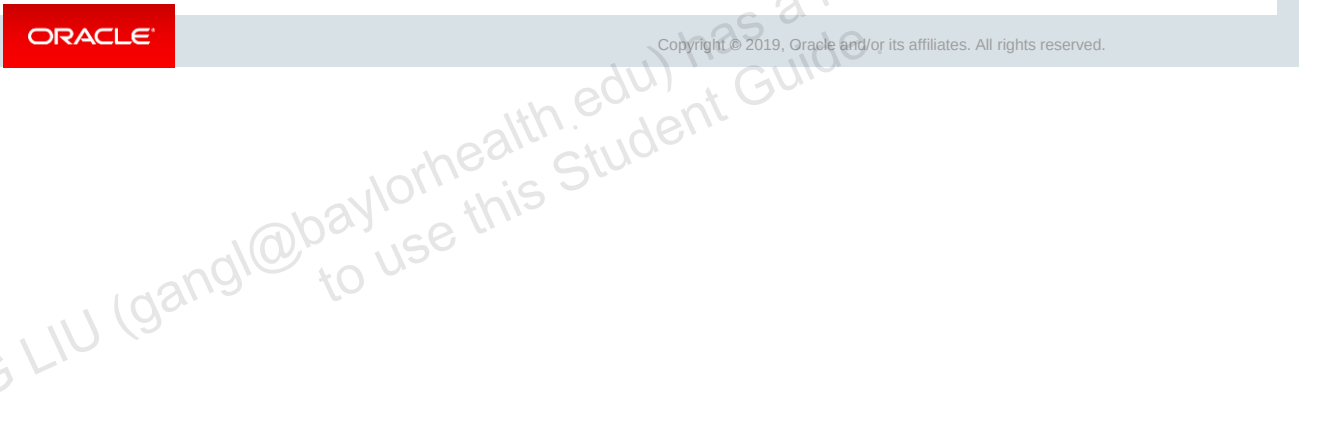
1. Connect to a read/write instance.
2. Execute **`cluster.dissolve({ force: true })`**:
   - Removes all cluster metadata and configuration and disables group replication
   - Data that was replicated between the instances remains intact
   - Nullifies any variables that were assigned to the `Cluster` object

```
mysql-js> \connect 'root@localhost:3310'
mysql-js> var cluster = dba.getCluster("mycluster")
mysql-js> cluster.dissolve({force:true})
The cluster was successfully dissolved.
Replication was disabled but user data was left intact.
```

Summary

In this lesson, you should have learned how to:

- Describe MySQL InnoDB Cluster and Group Replication
- List typical use cases for MySQL InnoDB Cluster
- Contrast the two different modes for deployment
- Configure a MySQL InnoDB Cluster
- Administer a MySQL InnoDB Cluster



Practices

- 14-1: Creating Member Instances
- 14-2: Creating MySQL InnoDB Cluster
- 14-3: Deploying MySQL Router and Testing the Cluster
- 14-4: Testing High Availability

Conclusion

15

Course Goals

In this course, you should have learned to:

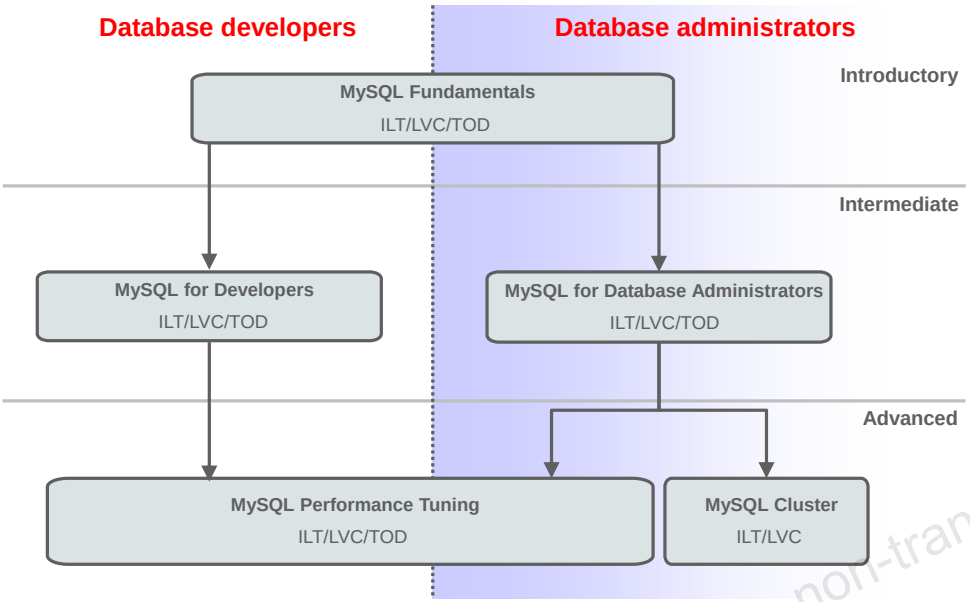
- Describe MySQL products and services
- Access MySQL resources
- Install the MySQL server and client programs
- Upgrade MySQL on a running server
- Describe MySQL architecture
- Explain how MySQL processes, stores, and transmits data
- Configure MySQL server and client programs
- Use server logs and other tools to monitor database activity
- Create and manage users and roles
- Protect your data from common security risks

Course Goals

In this course, you should have learned to:

- Maintain a stable system
- Troubleshoot server slowdowns and other common problems
- Identify and optimize poorly-performing queries
- Define and implement a backup strategy
- Perform physical and logical backups of your data
- Describe MySQL replication and its role in high availability and scalability
- Configure simple and complex replication topologies
- Administer a replication topology
- Configure and administer InnoDB Cluster

Oracle University: MySQL Training



MySQL Websites

- http://www.mysql.com includes:
  - Product information
  - Services (Training, Certification, Consulting, and Support)
  - White papers, webinars, and other resources
  - MySQL Enterprise Edition downloads (trial versions)
- http://dev.mysql.com includes:
  - Developer Zone (forums, articles, Planet MySQL, and more)
  - Documentation
  - Downloads
- https://github.com/mysql
  - Source code for MySQL Server and other MySQL products

Your Evaluation

- Courses are continually updated, and so your feedback is invaluable.
- Thank you for taking the time to give your opinions.

Thank You

- Congratulations on completing this course!
- Your attendance and participation are appreciated.
- For training and contact information, see the Oracle University website at http://www.oracle.com/education.

Q&A Session

- Questions and answers
- Questions after class
  - Get answers from the online reference manual at http://dev.mysql.com/doc/mysql/en/faqs.html.
- Example databases
  - Download the `world`, `employee`, and other sample databases from http://dev.mysql.com/doc/index-other.html.