
# Session 21

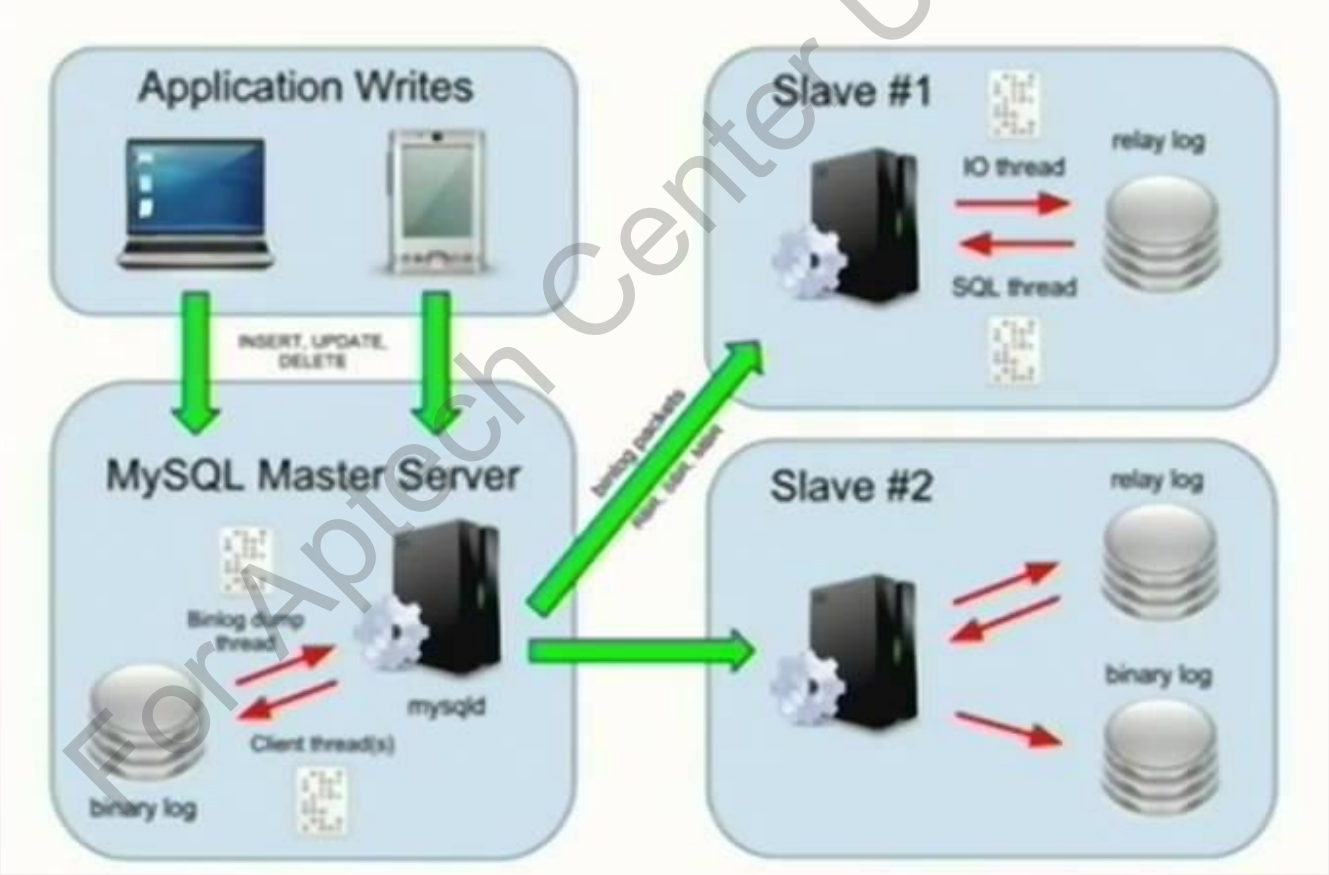
# Replication, Data Backup, Partitioning, and Storage Systems

# Objectives


- Explain replication in MySQL
- Explain how to manage data using replication
- Describe concepts of partitioning in MySQL
- Describe concepts of storage systems and management in MySQL

# Replication Basics

- Is asynchronous process of copying data from a MySQL server to single or multiple database servers.

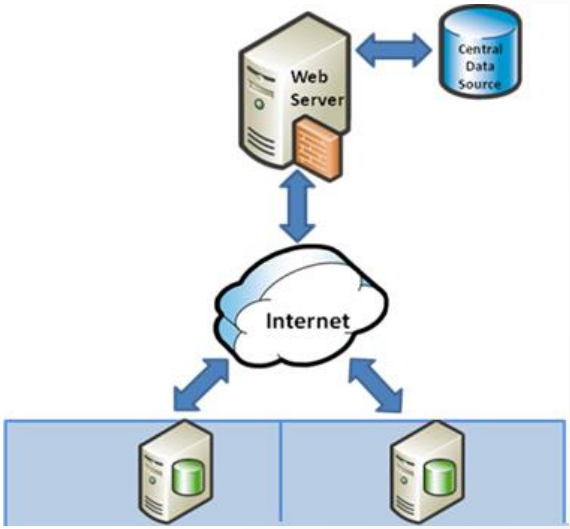


# Benefits of Replication

**Scaled Environment**

**Security**

**Independent Analytics**




**Remote Distribution**

# Methods of Replication


**Traditional**, which is based on binary log's replicating events to be synchronized between two servers

**Newer**, which is transactional and uses Global Transaction Identifiers (GTIDs) to ensures consistency

# Types of Synchronization


**Asynchronous**

- Involves a single server as the master and single or multiple other servers are slaves.

**Semisynchronous**

- Involves committing a transaction on the master only after a slave acknowledges it.

# Managing Replication


- It depends on the master's ability to track all changes across its databases in its binary log.
- The `SELECT` statements are not tracked.
- Each slave connecting to a master asks for a binary log copy for updating itself.
- Each slave is independent for making the updates at its own pace.

# Backup from Command Line 1-2


- Uses the `mysqldump` utility, which generates an SQL dump document after connecting to the MySQL server.
- Following is the syntax of using the utility:

```
$ mysqldump --opt -u [username] -
p[password] [databasename] > [dbbck.sql]
```

# Backup from Command Line 2-2


- Following code snippet shows the command to backup a database without a password:

```
$ mysqldump -u root -p School >
sch_backup.sql
```

- Following code snippet takes a backup of more than one database:

```
$ mysqldump -u root -p --databases School
College University > sch_backup.sql
```

# Backup via Compression 1-2


- Is ideal for taking a backup of a big MySQL database.
- Is done by compressing `mysqldump` output and piping it to gzip, which is a file compressor.

# Backup via Compression 2-2


- Following is the syntax for comrpessing the output:

```
$mysqldump -u [username] -p[password]
[databasename] | gzip -9 >
[bckfile.sql.gz]
```

- Following snippet shows how to extract a zipped file:

```
$ gunzip [bckupfile.sql.gz]
```

# Restoring a Database


- Following snippet restores a database without a password:

```
$ mysql -u root -p School < sch_backup.sql
```

- Following syntax restores a compressed backup file:

```
gunzip < [bckupfile.sql.gz] | mysql -u
[uname] -p[pass] [dbname]
```

# Partitioning Overview


- Lowers storage cost, boosts performance, and simplifying maintenance.
- Divides tables and indexes into smaller pieces across a file system.
- Allows queries to access only the desired part of records for quicker scan and faster results.
- Is of two types:
  - Horizontal
  - Vertical

# Partitioning in MySQL


- Is not of vertical type.
- Involves a user-defined rule called partitioning function, which accepts a parameter as user-stated expression.
- Involves an expression, which can be a column value or values.
- Is supported or not supported in MySQL, which is shown by the output of the `SHOW PLUGINS` statement.

# RANGE Partitioning


- Allows splitting a table as per different ranges of values.
- Following code snippet splits a table into four partitions, each by a range of id values:

```
CREATE TABLE emp(
 id int(11) NOT NULL,
 firstname varchar(255) NOT NULL,
 lastname varchar(255) NOT NULL)
 PARTITION BY RANGE (id) (
 PARTITION p0 VALUES LESS THAN (1000),
 PARTITION p1 VALUES LESS THAN (3000),
 PARTITION p2 VALUES LESS THAN (10000),
 PARTITION p3 VALUES LESS THAN (15000) );
```

# RANGE Partition Status


- Following figure shows the partition status of a table by querying `INFORMATION_SCHEMA.PARTITIONS`:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME = 'EMP';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |        999 |
| p1             |       2000 |
| p2             |       7000 |
| p3             |          0 |
+----------------+------------+
4 rows in set (0.00 sec)
```

# Dropping a RANGE Partition


- Following figure shows how to drop a partition and delete all rows from it using `TRUNCATE PARTITION`:

```
mysql> ALTER TABLE EMP TRUNCATE PARTITION P2;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT COUNT(*) FROM EMP;
+----------+
| COUNT(*) |
+----------+
|     2999 |
+----------+
1 row in set (0.00 sec)
```

# LIST Partitioning


- Allows splitting a table as per the pre-defined set of values.

- Following code snippet splits a table into three partitions, each by a list of id values:

```
CREATE TABLE emp_list(
 IDNo int(11) NOT NULL,
 firstname varchar(255) NOT NULL,
 lastname varchar(255) NOT NULL)
 PARTITION BY LIST (IDNo) (
 PARTITION partA VALUES IN (1,3,5),
 PARTITION partB VALUES IN (2,9,7),
 PARTITION partC VALUES IN (4,6,8));
```

# COLUMNS Partitioning 1-2


- Allows splitting a table by referring to more than one column.
- Are of two types namely, RANGE and COLUMNS, both accepting integer, date, and string types but not FLOAT, DECIMAL, BLOB, and TEXT types.

# COLUMNS Partitioning 2-2


- Following code snippet splits a table using RANGE COLUMNS partitioning:

```
CREATE TABLE DATA_COL_RANGE(
 id int(11) NOT NULL,
 numericdata int(11) NOT NULL,
 stringdata varchar(255) NOT NULL,
 comment varchar(255) NOT NULL)
 PARTITION BY RANGE COLUMNS (id, numericdata,
  stringdata) (
 PARTITION pn0 VALUES LESS THAN (100, 150, 'bbbb'),
 PARTITION pn1 VALUES LESS THAN (150, 250,'yyyy'),
 PARTITION pn2 VALUES LESS THAN (MAXVALUE,
  MAXVALUE, MAXVALUE));
```

# HASH Partitioning


- Splits the table across the specified number of partitions, according to a column value or expression to be hashed.
- Removes the need of stating the partitions explicty.
- Following snippet splits the table by hashing on the ID:

```
CREATE TABLE emp_hash(
 id int(11) NOT NULL,
 firstname varchar(255) NOT NULL,
 lastname varchar(255) NOT NULL))
 PARTITION BY Hash (id)
 PARTITIONS 4;
```

# KEY Partitioning


- Involves the server that offers its own hashing function for partitioning.
- Accepts table's primary or unique key as the partitioning key.
- Following snippet splits the table usingby using its primary key:

```
CREATE TABLE ITemp (IDNo INT NOT NULL PRIMARY
  KEY,
firstname VARCHAR(25), lastname VARCHAR(25))
PARTITION BY KEY()
PARTITIONS 2;
```

# SUB Partitioning


- Splits further each partition of a partitioned table.
- Following snippet further splits a partitioned table:

```
CREATE TABLE emp_SUB_PAR(
 id int(11) NOT NULL,
 firstname varchar(255) NOT NULL,
 lastname varchar(255) NOT NULL)
 PARTITION BY RANGE (id)
 SUBPARTITION BY hash (id)
 SUBPARTITIONS 4 (
 PARTITION p0 VALUES LESS THAN (1000),
 PARTITION p1 VALUES LESS THAN (3000),
 PARTITION p2 VALUES LESS THAN (10000),
 PARTITION p3 VALUES LESS THAN MAXVALUE );
```

# Managing HASH Partitions


- Following snippet reduces partitions of a table from 11 to 8:

```
CREATE TABLE customers (
 ID INT,
firstname VARCHAR(40),
lastname VARCHAR(40),
signed DATE )
PARTITION BY HASH( MONTH(signed) )
 PARTITIONS 11;
ALTER TABLE clients COALESCE PARTITION 3;
```

# Managing Partitions


Rebuilding

Optimizing

Analyzing

Repairing


Checking

# Storage Engines


- Refer to software components for creating, reading, storing, and updating the data in a database.
- Are of two types namely, transactional and non-transactional.
- Employ a pluggable architecture that loads and unloads engines from the server.
- Are viewable through the `SHOW ENGINES` statement.

# Storage Engines in MySQL 1-2


- InnoDB (default)
- MyISAM
- Memory
- CSV
- Archive
- Blackhole

# Storage Engines in MySQL 2-2


- NDB
- Merge
- Federated
- Example

# Using a Storage Engine


- Following code snippet shows how to specify a storage engine for a table:

```
CREATE TABLE CarModels(MID INTEGER PRIMARY KEY,
Name VARCHAR(60), Price INTEGER) ENGINE='MyISAM';
```

- Following code snippet shows how to default storage engine:

```
SHOW VARIABLES LIKE 'storage_engine';
```

# Transaction Model 1- 2


- InnoDB transaction model merges the conventional 2-Phase Locking (2PL) ability with a multi-versioning database.
- All user actions happen within a transaction, which is either rolled back or committed.
- InnoDB supports four isolation levels namely:
  - `REPEATABLE READ` (default)
  - `SERIALIZABLE`
  - `READ UNCOMMITTED`
  - `READ COMMITTED`

# Transaction Model 2- 2


- It is possible to alter the isolation level for a session or all succeeding sessions by using the `SET TRANSACTION` statement.
- Locking is performed on the row level and it executes queries by default as non-locking consistent reads.
- InnoDB employs next-key locking mechanism for row-level locking.

# Summary 1-2


- Replication refers to the process of copying data from a MySQL master server to single or multiple slave servers.
- Traditional replication is based on events of master's binary log.
- Transactional replication is dependent upon GTIDs, providing to be a simpler method.
- You can backup and restore a database by using the `mysqldump` utility, compressing its output, or via PHPMyAdmin tool.
- Partitioning divides tables and indexes into smaller pieces for decreasing the cost of data storage, enhancing performance, and simplifying maintenance.
- A user-defined rule for splitting the table data is termed as partitioning function, which can be a hashing function, modulus, or a match for a list or range.

# Summary 2-2


- The `SHOW PLUGINS` statement allows checking whether or not the current server instance supports partitioning.
- You can partition a table by ranges, lists, hash function, columns, and keys.
- You can rebuild, optimize, analyze, check, and fix partitions.
- InnoDB is the default general-purpose storage engine for version MySQL 5.7.
- MySQL 5.7 supports different storage engines such as `MyISAM`, `CSV`, `NDB`, `Archive`, `Merge` `Memory`, `Federated`, and `Example`.
- The default isolation level of InnoDB is `REPEATABLE READ`.