Database Management Systems Training

June 12 - 16, 2023

One Central Hotel
Cebu City, Philippines

SHOW Command

SHOW Command

- It is specifically used to retrieve information about database objects or metadata.
- It allows you to view details about databases, tables, columns, indexes, users, privileges, and other aspects of the database schema.

Common Uses of the SHOW Command

SHOW DATABASES

- Lists all the databases available in the server
- Syntax:

```
SHOW DATABASES;
```

SHOW TABLES

- Lists all the tables in the current database or to the specified database.

- Syntax:

```
SHOW TABLES;
```

SHOW TABLES

Examples

```
-- list the tables of
-- the current database
SHOW TABLES;


-- list the tables of
-- the testdb database
SHOW TABLES FROM testdb;
```

SHOW COLUMNS

- Displays the columns and their properties for a specific table.
- Syntax:

```
SHOW COLUMNS FROM table_name;
```

SHOW COLUMNS

Examples

```
-- list the columns of
-- the employee table
-- of the current database
SHOW COLUMNS FROM employee;

-- list the columns of
-- the receipt table
-- of the testdb database
SHOW COLUMNS FROM testdb.receipt;
```

SHOW INDEX

- Shows the indexes defined on a table.
- Syntax:

```
SHOW INDEX FROM table_name;
```

SHOW INDEX

Examples

```
-- list the indexes of
-- the employee table
-- of the current database
SHOW INDEX FROM employee;

-- list the indexes of
-- the receipt table
-- of the testdb database
SHOW INDEX FROM testdb.receipt;
```

SHOW TABLE STATUS

- Provides information about a specific table, including its size, row count, and other details.

- Syntax:

```
SHOW TABLE STATUS [FROM database_name];
```

SHOW TABLE STATUS

Examples

```
-- table status of the current database
SHOW TABLE STATUS;

-- table status of the testdb database
SHOW TABLE STATUS FROM testdb;

-- table status of the testdb database
-- with added filter
SHOW TABLE STATUS FROM testdb LIKE 'a%';
```

SHOW CREATE TABLE

- Displays the SQL statement used to create a specific table.
- Syntax:

```
SHOW CREATE TABLE table_name;
```

SHOW CREATE TABLE

Examples

```
-- show DDL info of the employee table
SHOW CREATE TABLE employee;

-- show DDL info of the receipt table
-- in the testdb database
SHOW CREATE TABLE testdb.receipt;
```

SHOW GRANTS

- Lists the privileges and permissions granted to a specific user.
- Syntax:

```
SHOW GRANTS FOR user_name;
```

SHOW GRANTS

Examples

```
-- show permissions granted
-- to the current user
SHOW GRANTS;

-- show permissions granted
-- to the admin_user
SHOW GRANTS FOR admin_user;
```

SHOW VARIABLES

- Displays the current values of server variables and their settings.
- Syntax:

```
SHOW VARIABLES;
```

SHOW VARIABLES

Examples

```
-- show all variables
SHOW VARIABLES;

-- show variables with filter
SHOW VARIABLES LIKE 'max%';
```

SHOW STATUS

- Provides various runtime information and statistics about the server.

- Syntax:

```
SHOW STATUS;
```

SHOW STATUS

Examples

```
-- display all status
SHOW STATUS;

-- display status with filter
SHOW STATUS LIKE 'max%';
```

SHOW ENGINE STATUS

- Displays specific information about a storage engine.
- Syntax:

```
SHOW ENGINE engine_name STATUS;
```

SHOW ENGINE STATUS

Examples

```
SHOW ENGINE INNODB STATUS;
```

SHOW OPEN TABLES

- Lists the currently open tables in the server.
- Syntax:

```
SHOW OPEN TABLES;
```

SHOW PROCESSLIST

- Displays the currently running processes or queries.
- Syntax:

```
SHOW PROCESSLIST;
```

SQL Functions

SQL Functions

- SQL functions are built-in operations that perform specific tasks on data or manipulate the data in some way.
- They can be used to retrieve, transform, calculate, or manipulate data within SQL queries.

Commonly used SQL functions

- Aggregate Functions
  - SUM(): Calculates the sum of values in a column.
  - AVG(): Calculates the average of values in a column.
  - COUNT(): Counts the number of rows or non-null values in a column.
  - MIN(): Retrieves the minimum value from a column.
  - MAX(): Retrieves the maximum value from a column.

String Functions

- **CONCAT**(): Concatenates two or more strings together.
- **UPPER**(): Converts a string to uppercase.
- **LOWER**(): Converts a string to lowercase.
- **LENGTH**(): Retrieves the length of a string.
- **SUBSTRING**(): Extracts a portion of a string.

Date and Time Functions

- **NOW**(): Retrieves the current date and time.
- **DATE**(): Extracts the date part from a datetime value.
- **TIME**(): Extracts the time part from a datetime value.
- **YEAR**(): Extracts the year from a date value.
- **MONTH**(): Extracts the month from a date value.
- **DAY**(): Extracts the day from a date value.

Numeric Functions

- **ABS**(): Returns the absolute value of a number.
- **ROUND**(): Rounds a number to a specified decimal place.
- **CEILING**(): Rounds a number up to the nearest integer.
- **FLOOR**(): Rounds a number down to the nearest integer.
- **POWER**(): Raises a number to a specified power.

Conditional Functions

- **CASE WHEN**:
  - Performs conditional logic to return different values based on specified conditions.
- **COALESCE**():
  - Returns the first non-null value from a list of expressions.

SQL Functions

For more details of the other functions please visit the following:

- https://www.w3schools.com/mysql/mysql_ref_functions.asp

SQL Alias

AS keyword

- Alias is used to give an alias name to a table or a column, which can be a result-set table too.
- This is quite useful in case of large or complex queries.
- Alias is mainly used for giving a short alias name for a column or a table with complex names.

AS keyword

Examples

```
SELECT
    r.receiptdate,
    r.receiptno,
    r.amount,
    NOW() as rundate
FROM receipt r
;
```

SQL SET Operations

SET Operations

- SET operations are used to combine or manipulate the results of multiple SELECT statements.
- They allow you to perform operations such as union, intersection, and difference on the result sets of two or more SELECT statements.

UNION

- Combines the result sets of two or more SELECT statements into a single result set, removing duplicate rows.

UNION

Syntax

```
SELECT columns
FROM table1

UNION

SELECT columns
FROM table2

;
```

UNION ALL

- Combines the result sets of two or more SELECT statements into a single result set, including duplicate rows.

UNION ALL

Syntax

```
SELECT columns
FROM table1

UNION ALL

SELECT columns
FROM table2

;
```

INTERSECT

- Retrieves the common rows that appear in the result sets of two or more SELECT statements.

INTERSECT

Syntax

```
SELECT columns
FROM table1

INTERSECT

SELECT columns
FROM table2

;
```

SQL Subquery

Subquery

- A subquery is a SQL query nested inside a larger query.
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- Subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.

Subquery

- A subquery is usually added within the WHERE clause of another SQL SELECT statement.
- A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
- The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query.

Subquery

Example

```
SELECT
    c.receiptdate,
    COUNT(c.objid) as totalcount,
    SUM(c.amount) as totalamt,
    SUM((
        SELECT c.amount
        FROM cashreceipt_void
        WHERE receiptid = c.objid
    )) as totalvoidamt
FROM cashreceipt c
GROUP BY c.receiptdate
;
```

SQL Join

SQL Join

- It is a clause used to combine rows from two or more tables based on related columns between them.
- It allows you to retrieve data from multiple tables in a single query by establishing relationships between the tables.
- A table can also join to itself, which is known as, **Self Join**.

INNER JOIN

- Retrieves rows that have matching values in both tables being joined.

- Syntax:

```
SELECT columns
FROM table_1 t1
    INNER JOIN table_2 t2 ON t2.column = t1.column
;
```

INNER JOIN

Examples

```
SELECT
    r.receiptdate, r.receiptno, ri.*
FROM receipt r
    INNER JOIN receiptitem ri ON ri.receiptid = r.objid
;
```

LEFT JOIN

- Retrieves all rows from the left table and the matching rows from the right table.

- Syntax:

```
SELECT columns
FROM table_1 t1
    LEFT JOIN table_2 t2 ON t2.column = t1.column
;
```

LEFT JOIN

Examples

```
SELECT
    r.receiptdate, r.receiptno, ri.*
FROM receipt r
    LEFT JOIN receiptitem ri ON ri.receiptid = r.objid
;
```

RIGHT JOIN

- Retrieves all rows from the right table and the matching rows from the left table.

- Syntax:

```
SELECT columns
FROM table_1 t1
    RIGHT JOIN table_2 t2 ON t2.column = t1.column
;
```

FULL JOIN

- Retrieves all rows from both tables and includes the unmatched rows as well.

- Syntax:

```
SELECT columns
FROM table_1 t1
    FULL JOIN table_2 t2 ON t2.column = t1.column
;
```

CROSS JOIN

- Retrieves the Cartesian product of the two tables, resulting in all possible combinations of rows.

- Syntax:

```
SELECT columns
FROM table_1 t1
    CROSS JOIN table_2 t2
;
```

Introduction to Database Administration

Database Administration

- Database administration involves managing and maintaining a database system to ensure its efficient and reliable operation.
- It encompasses various tasks related to the installation, configuration, monitoring, backup, recovery, security, and performance optimization of databases.

Database Administration Tasks

1. Installation and Configuration

   - Installing and configuring the database management system (DBMS) software on servers or machines.

   - Setting up database instances and configuring system parameters.

Database Administration Tasks

2. User and Security Management

   - Creating and managing user accounts with appropriate privileges.
   - Enforcing data security and access controls to protect sensitive information.
   - Implementing authentication and authorization mechanisms.

Database Administration Tasks

3. Database Design and Schema Management

   - Collaborating with application developers to design efficient database schemas.
   - Creating and managing database objects like tables, views, indexes, and constraints.
   - Ensuring data integrity and maintaining referential integrity through constraints.

Database Administration Tasks

4. Performance Monitoring and Optimization

   - Monitoring and analyzing database performance to identify bottlenecks or areas for improvement.
   - Tuning database configurations, indexes, and queries to enhance performance.
   - Implementing indexing strategies and optimizing query execution plans.

Database Administration Tasks

5. Backup and Recovery

   - Planning and implementing backup and recovery strategies to safeguard data against loss or corruption.
   - Performing regular database backups and ensuring their integrity.
   - Developing and testing recovery procedures to restore data in case of failures.

Database Administration Tasks

6. Database Maintenance

   - Monitoring and managing database storage, including space allocation and growth.
   - Performing database maintenance tasks such as reorganizing indexes, updating statistics, and optimizing storage.
   - Managing database upgrades, patches, and software updates.

Database Administration Tasks

7. Disaster Recovery and High Availability

   - Implementing disaster recovery plans to minimize downtime and data loss in the event of system failures.
   - Configuring high availability solutions like database replication, clustering, or failover mechanisms.
   - Conducting periodic disaster recovery drills and testing backup and recovery procedures.

User Management / Access Control

User Management / Access Control

- User management and access control are essential aspects of database administration.
- They involve creating and managing user accounts, assigning appropriate privileges, and enforcing security measures to control access to the database system and its data.

1. User Account Creation

- Creating user accounts for individuals or applications that need access to the database.
- Assigning a unique username and password for each user account.
- Specifying other account details, such as account expiration, default schema, or session settings.

2. Privilege Assignment

- Granting appropriate privileges to user accounts to define their level of access to the database.
- Privileges include read (SELECT), write (INSERT, UPDATE, DELETE), and administrative permissions.
- Privileges can be granted at the database, schema, table, or column level, allowing fine-grained control.

3. Role-Based Access Control (RBAC)

- Creating user roles that group together common sets of privileges.
- Assigning roles to user accounts instead of assigning privileges individually.
- Simplifies user management and ensures consistent access control across multiple users.

4. Authorization and Authentication

- Implementing authentication mechanisms to verify the identity of users accessing the database.
- Common authentication methods include username/password authentication, LDAP integration, or single sign-on (SSO).
- Enforcing authorization rules to verify that users have the necessary privileges to perform requested actions.

5. Access Control Policies

- Defining and enforcing access control policies based on business requirements and regulatory compliance.
- Implementing mechanisms to restrict access to sensitive data, such as personally identifiable information (PII) or financial data.
- Applying row-level or column-level security measures to control data visibility based on user roles or attributes.

6. Audit and Monitoring

- Monitoring user activities and database events to detect unauthorized access or suspicious behavior.
- Logging and reviewing database activity logs for security and compliance purposes.
- Implementing intrusion detection systems (IDS) or database activity monitoring (DAM) tools to enhance security.

Backup and Restore

Backup and Restore

- Backup and Restore are critical operations in database management to protect data from loss, corruption, or accidental deletion.
- It is essential to carefully plan and document the backup and restore process, including backup strategies, schedules, retention policies, and disaster recovery plans.
- Regularly validate and monitor the backup integrity to ensure data recoverability.
- Database administrators should also consider automation tools and technologies to streamline the backup and restore process and ensure data availability and business continuity.

Backup and Restore Process - Overview

1. Backup Types

- **Full Backup**: Creates a complete copy of the entire database, including all data and objects.
- **Incremental Backup**: Captures only the changes made since the last backup, reducing backup time and storage requirements.
- **Differential Backup**: Backs up the changes made since the last full backup, providing faster restore times compared to incremental backups.

2. Backup Methods

- **Physical Backup**: Copies the physical database files directly, such as data files, log files, and control files.
- **Logical Backup**: Exports the logical data and database structure using SQL statements (e.g., INSERT, CREATE).
- **Online Backup**: Performed while the database is running and accepting user transactions.
- **Offline Backup**: Requires the database to be offline during the backup process.

3. Backup Schedule

- Establish a backup schedule based on the criticality of the data and the frequency of updates.
- Regularly perform full backups, supplemented by incremental or differential backups for efficiency.
- Consider retention policies to determine how long backups should be kept.

4. Restore Process

- To restore a database, first ensure that backups are available and valid.
- Restore the most recent full backup and apply subsequent incremental or differential backups, if applicable.
- Perform necessary recovery steps, such as applying transaction logs, to bring the database to a consistent state.

5. Testing and Verification

- Periodically test the backup and restore process to ensure the backups are valid and can be successfully restored.
- Verify the integrity and consistency of restored databases by performing data validation and running test queries.

6. Off-site Storage and Disaster Recovery

- Store backups in a secure off-site location to protect against local hardware failures, disasters, or data breaches.
- Implement a disaster recovery plan to outline procedures for recovering the database in case of a catastrophic event.

Backup Database in MySQL

Backup Database

- In MySQL, the commonly used command for performing a backup is **mysqldump**
- **mysqldump** is a command-line utility that is used for creating logical backups of MySQL databases.
- It allows you to generate a text file containing the SQL statements needed to recreate the database structure (*including tables, views, procedures, etc.*) and data.

Backup Database

Syntax

```
mysqldump
    --max-allowed-packet=16M
    -u user_name -p -f
    database_name > backup_file.sql
```

Restore Database in MySQL

1. Create an Empty Database

- If you want to restore the backup into a new database, you can create an empty database using the following command:

```
mysql
    -u user_name -p -e
    "CREATE DATABASE new_database_name"
```

  - Replace **user_name** with your MySQL username and **new_database_name** with the desired name for the new database.

2. Choose the Backup Method

- If you have a backup file generated by **mysqldump**, you can restore it using the following command:

```
mysql
    -u user_name -p -f
    -D database_name < backup_file.sql
```

- Replace **user_name** with your MySQL username, **database_name** with the name of the database you want to restore into, and **backup_file.sql** with the path to your backup file.

3. Enter the MySQL Password

- After executing the restore command, you will be prompted to enter the password for your MySQL user.
- Enter the password and press Enter.

4. Wait for the Restore to Complete

- The restore process will start, and MySQL will execute the SQL statements in the backup file, recreating the database structure and inserting the data.

- Wait for the process to complete, which may take some time depending on the size of the database.

5. Verify the Restore

- Once the restore process finishes, you can verify the restored database by connecting to MySQL and querying the data or inspecting the database using administration tools.