* DROP command in SQL

The DROP command in SQL is used to delete database objects like tables, database, views, indexes or ~~she~~ schemas. When you use DROP, it completely removes the object and its data from the database, unlike DELETE, which removes data but leaves the table structure intact, DROP removes both the Data and the structure.

1. DROPPING a Table

Syntax :- DROP TABLE table_name;

Example :- DROP TABLE Employees;

Note :- This will delete the Employees table along with all its data

2. DROPING A Database

Syntax :- DROP DATABASE database_name;

Example :- DROP DATABASE companyDB;

Note :- This will delete the entire CompanyDB database, including all tables and data within it.

3. Droping a view

Syntax :- DROP VIEW view_name;

Example :- DROP ~~VIEW~~ VIEW HighSalaryEmployees;

Note :- This will remove the view HighSalaryEmployees.

4- DROPING an Index

Syntax :- DROP INDEX index_name ON table_name;

Example:- DROP INDEX idx_name ON Employees;

Note:- This will remove the idx_name index on the Employees table.

5. Dropping a Schema

Syntax:- DROP SCHEMA Schema_name;

example:- DROP SCHEMA SalesData;

Note:- The DROP command is immediate and cannot be rolled back, so it should be used with caution.

---

* RENAME Command in SQL

The RENAME command is SQL is used to change the name of a database object, such as a table or column. Depending on the SQL dialect RENAME may have slightly different syntax, but it commonly involves renaming tables or columns within a table.

* RENAME TABLE and COLUMN as same as ALTER RENAME.

Alternative Syntax for Renaming Tables (for mySQL)

Syntax:- RENAME TABLE old_table_name TO new_table_name;

Example:- RENAME TABLE Employees TO staff;

Caution:- Renaming columns and tables is often permanent and can impact applications or queries that depend on the original names.

* TRUNCATE COMMAND IN SQL

01/11/2024
12:20am

TRUNCATE

The TRUNCATE command in mySQL is used to quickly delete all rows from a table while keeping the table structure intact. This is more efficient than using DELETE without a WHERE clause because TRUNCATE performs the operation without logging individual row deletions; which can be faster, especially for large tables.

Syntax: TRUNCATE TABLE table_name

Example: TRUNCATE TABLE Employees;

Note: This command will delete all rocws in the Employees table, but the table itself along with its structure, columns and Indexes will remain.

---

* DATA MANIPULATION LANGUAGE (DML)

In mySQL DML includes SQL commands used to manipulate data within existing tables. These command focus on querying, Inserting, updating, and deleting data but don't affect the structure of the tables themselves.

DML commands in mySQL - INSERT, UPDATE, DELETE, SELECT

1. INSERT command: The INSERT command in mySQL is used to add new rows of data into a existing table;

Basic Syntax: INSERT INTO table_name (col1, col2, col3, ----)
VALUS (val1, value-2, value-3, ---- -);

* Variations of INSERT Command

1. Inserting All columns (without specifying column names)

If we are adding values to all columns in the order the are defined in the table, we can skip the column names!

Example of SQL query

INSERT INTO table_name VALUES (val-1, val 2, val 3, ------);

⇒ CREATE DATABASE company;

USE company;

CREATE TABLE Information (

Id INT primary KEY, Name VARCHAR(50), Age INT);

Insert values in rows with columns name and values name

INSERT INTO Information (Id, Name, Age)
values (1, "Ramakant", 21)

Insert values without using columns name.

INSERT INTO Information Values (2, "Arvind", 22);

2- Inserting Data into specific columns : If we want to Insert values into only specific columns, we must list those columns explicitly:

INSERT INTO table_name (column1, column3)
VALUES (value-1, value-3);

⇒ INSERT INTO Information (Id, Age)
VALUES (3, 21);

31/7/2024

* SELECT COMMAND

The select command in SQL is one of the most fundamental and widely used commands. It is part of DQL (Data Query language), a subset of SQL that deals specifically with querying data from the database. The select command allows users to retrieve specific data from one or more tables, perform calculations and aggregate data to gain insights.

Basic Structure :-

```
SELECT column1, column2, .....
FROM table_name
WHERE condition
GROUP BY Column
HAVING condition
ORDER BY Column ASC/DESC
LIMIT number;
```

Each part of this syntax provides specific functionality

1. SELECT clause

The SELECT clause specifies the columns to retrieve data from. We can choose one or multiple columns use expressions, and even functions.

Example

- * SELECT all columns (table)

```
SELECT * FROM employees;
```

The * symbol selects all columns from the Employees table

* SELECT specific columns:

SELECT first_name, last_name FROM employees;

The command retrieves only the first_name and last_name columns from the employees table.

2. FROM Clause

The FROM clause specifies the table(s) from which to retrieve data. It is a required part of the SELECT statement, as it defines where the data originates.

Eg:

SELECT first_name FROM employees;

Here, the employees table is specified as the source of data for the first_name column.

3. WHERE Clause

The WHERE clause filters records based on specified conditions, allowing you to retrieve only the rows that meet certain criteria.

SELECT first_name, lastname FROM employees WHERE department = 'Sales';

This query retrieves data only for employees who work in the Sales.

* Common comparison operators in WHERE:

* =: Equals
* <> or !=: Not equals
* > / <: Greater than / less than
* >= / <=: Greater than or equal to / Less than or equal to
* BETWEEN: Within a specified range

* LIKE: Pattern matching

* IN: Matches any values in a list.

* (4) GROUP BY Clause:

The GROUP BY clause groups rows with the same values in specified columns into summary rows, often used with aggregate functions (like count, Sum, AVG..etc.)

Example:

```
SELECT department, COUNT(*) AS total_employees
  FROM employees GROUP BY department;
```

The query counts the total number of employees in each department.

* (5) HAVING clause

The HAVING clause filters groups created by the GROUP BY clause based on specific conditions. Unlike WHERE, which filters individual rows, HAVING filters aggregate data.

Example:

```
SELECT department, COUNT(*) AS total_employees.
  FROM employees,
GROUP By department
  HAVING COUNT(*) >10;
```

This query returns department with more than 10 employees.

(6) ORDER BY Clause:

The ORDER BY clause sorts the results based on one or more columns. By default, it sorts in ascending order (ASC), but we can specify descending order (DESC) as well

Example:

```
SELECT first_name, Salary FROM employees ORDER BY
Salary DESC
```

The query retrieves the first names and salaries of employees, sorted by Salary in descending order.

(7) LIMIT Clause:

The LIMIT clause restricts the numbers of rows returned by a query. It is often used when only a subset of result is needed

Example:

```
SELECT first_name, last_name FROM employees
LIMIT 5;
```

This query retrieves the first five rows from the employees table.

* ADVANCED FEATURES in SELECT

(1) Aliasing of columns and Tables.

Aliasing ~~rem~~ rename columns or tables temporarily to simplify queries or improve readability.

Example:

```
SELECT first_name AS fname, last_name AS lname
FROM employees AS emp;
```

Here first_name and last_name are renamed to fname and lname and employees is temporarily renamed emp.

Using functions in SELECT

SQL includes various functions that can be used within SELECT statements

* Aggregate function.
* COUNT( ): counts rows
* SUM(): calculates the sum
* AVG(): Calculates the Average
* MAX(): Returns the maximum value.
* MIN(): Returns the minimum value.

Eg:

```
SELECT COUNT(*) AS total_employees
FROM employees;
```

* String Functions (e.g. UPPER( ), LOWER( ), CONCAT(

Example:

```
SELECT CONCAT(first_name, ' ', last_name) AS
full_name FROM employees;
```

* Date Functions (eg- NOW( ), YEAR( ), MONTH()

```
SELECT first_name, YEAR(hire_date) AS hire_year
FROM employees;
```

(3) Using Subqueries

A Subquery is a query nested within another SQL query, It is used for more advanced filtering and data extraction.

Example:

```
SELECT first_name FROM employees
WHERE salary > (SELECT AVG(salary) FROM
                employees);
```

This query retrieves employees whose salaries are above the average salary.

* OFFSET :- The OFFSET clause in SQL is used to skip a specified number of rows before stating to return rows in the result set. This is often used in conjunction with the limit (LIMIT) clause to implement Pagination which is useful for displaying data in chunks or pages on applications or websites. which is useful for displaying data in chunks or pages on applications or websites.

Syntax of OFFSET :-

The Syntax for OFFSET varies slightly depending on the SQL database, but the general format is.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column ASC/DESC
LIMIT row_count OR OFFSET offset_value;
```

→ LIMIT row_count : Specifies the maximum number of rows to return.

→ OFFSET offset_value : Specifies the number of rows to skip Before starting to return rows.

Example of OFFSET with LIMIT

Consider a table called employees with multiple rows, we use OFFSET like this

```
SELECT first_name, last_name
FROM employees
ORDER BY employee_id
LIMIT 5 OFFSET 10;
```

⇒ The query will skip the first 10 rows and then retrieve the next 5 rows.

⇒ The rows are sorted by employee_id, ensuring consistent ordering

* Common use case: PAGINATION

When displaying data in pages, we can use OFFSET and LIMIT together to get different "Pages" of data. For example. If we want to display 10 result per page.

1. Page-1

```
SELECT * FROM employees ORDER BY employee_id
LIMIT 10 OFFSET 0;
```

This return 1-10

2 Page-2

```
SELECT * FROM employees ORDER BY employee_id
LIMIT 10 OFFSET 10;
```

This skips the first 10 rows and return rows 11-20

* Important Notes:-

* Ordering + OFFSET Should generally be used with ORDER BY to ensure consistent

results, as rows can be returned in any order without a sorting.

* PERFORMANCE :- In large datasets; high OFFSET values can impact performance since the database engine still scans skipped rows

* DISTINCT :- The DISTINCT keyword in SQL is used to remove duplicate values from the result set of a SELECT query. When ~~you are~~ we use DISTINCT, SQL returns only unique values in the specified columns. This is especially useful when you want to get a list of unique entries from a column that may contain duplicates.

* Basic Syntax :-

```
SELECT DISTINCT column1, column2, ....
FROM table_name;
```

* HOW DISTINCT WORKS

⇒ DISTINCT applies to the entire ~~row~~ row, not just a single column, so if you select multiple columns with DISTINCT, SQL will return unique combinations of those columns.

⇒ This means if we specify multiple columns with DISTINCT SQL treats each combination of those columns as unique and will only ~~return~~ return distinct combinations.

Example: Consider a table called employees with the following data.