MySQL Database Keywords and Examples

Data Definition Language (DDL) Keywords

CREATE: Creates a new database, table, or object.

```
CREATE DATABASE employee;

CREATE TABLE employees (employee_id INT PRIMARY KEY, first_name VARCHAR(50), last_name  VARCHAR(50),
job_title VARCHAR(50));
```

DROP: Deletes a database, table, or object.

```
DROP TABLE employees;

DROP DATABASE employee;
```

ALTER: Modifies an existing table or database schema.

```
ALTER TABLE employees ADD COLUMN email VARCHAR(100);

ALTER TABLE employees MODIFY COLUMN job_title VARCHAR(100);
```

TRUNCATE: Removes all rows from a table.

```
TRUNCATE TABLE employees;
```

RENAME: Renames a table or database.

```
RENAME TABLE employees TO staff;
```

USE: Selects a database for subsequent queries.

```
USE employee;
```

Data Manipulation Language (DML) Keywords

SELECT: Retrieves data from one or more tables.

```
SELECT * FROM employees;

SELECT first_name, last_name FROM employees;
```

INSERT: Adds new rows to a table.

```
INSERT INTO employees (employee_id, first_name, last_name, job_title) VALUES (1, 'John', 'Doe', 'Software
Engineer');
```

UPDATE: Modifies existing rows in a table.

```
UPDATE employees SET job_title = 'Senior Software Engineer' WHERE employee_id = 1;
```

DELETE: Removes rows from a table.

```
DELETE FROM employees WHERE employee_id = 1;
```

REPLACE: Inserts or updates rows in a table.

```
REPLACE INTO employees (employee_id, first_name, last_name, job_title) VALUES (1, 'John', 'Doe', 'Lead
Developer');
```

Data Querying Keywords

WHERE: Filters rows based on a condition.

```
SELECT * FROM employees WHERE job_title = 'Software Engineer';
```

ORDER BY: Sorts rows in ascending or descending order.

```
SELECT * FROM employees ORDER BY last_name ASC;
```

GROUP BY: Groups rows sharing a property.

```
SELECT job_title, COUNT(*) AS num_employees FROM employees GROUP BY job_title;
```

HAVING: Filters groups created by GROUP BY.

```
SELECT job_title, COUNT(*) AS num_employees FROM employees GROUP BY job_title HAVING COUNT(*) > 1;
```

LIMIT: Restricts the number of rows returned.

```
SELECT * FROM employees LIMIT 5;
```

MySQL Functions and Operators

Aggregate Functions

COUNT(): Counts rows.

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

SUM(): Calculates the sum of a column.

```
SELECT SUM(salary) FROM employees;
```

AVG(): Calculates the average of a column.

```
SELECT AVG(salary) FROM employees;
```

MIN(): Finds the minimum value in a column.

```
SELECT MIN(salary) FROM employees;
```

MAX(): Finds the maximum value in a column.

```
SELECT MAX(salary) FROM employees;
```

String Functions

CONCAT(): Joins two or more strings.

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

SUBSTRING(): Extracts a portion of a string.

```
SELECT SUBSTRING(first_name, 1, 3) FROM employees;
```

LENGTH(): Returns the length of a string.

```
SELECT LENGTH(last_name) FROM employees;
```

REPLACE(): Replaces occurrences of a substring.

```
SELECT REPLACE(job_title, 'Engineer', 'Developer') FROM employees;
```

MySQL Operators

=: Equals.

```
SELECT * FROM employees WHERE job_title = 'Software Engineer';
```

!= or <>: Not equals.

```
SELECT * FROM employees WHERE job_title != 'Software Engineer';
```

>: Greater than.

```
SELECT * FROM employees WHERE salary > 50000;
```

<: Less than.

```
SELECT * FROM employees WHERE salary < 50000;
```

LIKE: Pattern matching.

```
SELECT * FROM employees WHERE last_name LIKE 'D%';
```

IN: Matches any value in a list.

```
SELECT * FROM employees WHERE job_title IN ('Software Engineer', 'Manager');
```

BETWEEN: Checks if a value is within a range.

```
SELECT * FROM employees WHERE salary BETWEEN 40000 AND 70000;
```

Additional Keywords

Primary Key

Uniquely identifies each row in a table, ensuring no duplicates or NULL values. A table can have only one primary key.

Unique Key

Ensures all values in a column are distinct, allowing one NULL value. A table can have multiple unique keys.

Foreign Key

Links two tables by referencing the primary key of another table. Maintains referential integrity and enforces relationships.

NOT NULL:
Ensures a column cannot have NULL values.

Auto Increment is a database feature that automatically generates a unique value for a column whenever a new row is inserted. It is commonly used for primary key columns to ensure each record has a unique identifier.

Relationships:

- **One-to-One**: A single record in one table is linked to a single record in another.
- **One-to-Many**: A single record in one table is linked to multiple records in another.
- **Many-to-Many**: Records in one table are linked to multiple records in another and vice versa.

Steps to Apply a Foreign Key in MySQL

We have two tables:

1. **customers**: Contains customer details.
2. **orders**: Contains customer orders.

Step 1: Create the Parent Table (customers)

```
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

Step 2: Create the Child Table (orders) with a Foreign Key

```
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Key Points in the Code

1. **`FOREIGN KEY (customer_id) REFERENCES customers(customer_id)`**:
   - Specifies that the `customer_id` in the `orders` table references the `customer_id` in the `customers` table.
2. **`ON DELETE CASCADE`**:
   - If a customer is deleted, all their orders will also be deleted.
3. **`ON UPDATE CASCADE`**:
   - If the `customer_id` in the `customers` table is updated, it will automatically update in the `orders` table.

Advanced Join Queries on Customer and Order Tables

Schema Definitions

1. `customers` table

customer_id	name	email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
3	Mike Johnson	mike@example.com

2. `orders` table

order_id	order_date	customer_id	total_amount
101	2025-01-01	1	250.00
102	2025-01-02	2	300.00
103	2025-01-03	1	150.00
104	2025-01-04	4	200.00

1. INNER JOIN

The **INNER JOIN** operation extracts rows that exhibit congruence between the specified columns of both tables. This operation excludes non-matching rows from the result set.

```
SELECT
    c.customer_id,
    c.name AS customer_name,
    o.order_id,
    o.order_date,
    o.total_amount

FROM  customers c  INNER JOIN
    orders o ON c.customer_id = o.customer_id;
```

Result:

customer_id	customer_name	order_id	order_date	total_amount
1	John Doe	101	2025-01-01	250.00
1	John Doe	103	2025-01-03	150.00
2	Jane Smith	102	2025-01-02	300.00

2. LEFT JOIN

The **LEFT JOIN** operation incorporates all records from the left table (`customers`), supplementing them with corresponding rows from the right table (`orders`). Where no correspondence exists, `NULL` values populate the result set.

```
SELECT
    c.customer_id,
    c.name AS customer_name,
    o.order_id,
    o.order_date,
    o.total_amount
FROM
    customers c
LEFT JOIN
    orders o ON c.customer_id = o.customer_id;
```

Result:

customer_id	customer_name	order_id	order_date	total_amount
1	John Doe	101	2025-01-01	250.00
1	John Doe	103	2025-01-03	150.00
2	Jane Smith	102	2025-01-02	300.00
3	Mike Johnson	NULL	NULL	NULL

3. RIGHT JOIN

The **RIGHT JOIN** operation retrieves all rows from the right table (`orders`) and matches them with rows from the left table (`customers`). Rows from the right table without a corresponding match in the left table are supplemented with `NULL` values.

```
SELECT
    c.customer_id,
    c.name AS customer_name,
    o.order_id,
    o.order_date,
    o.total_amount
FROM
    customers c
RIGHT JOIN
    orders o ON c.customer_id = o.customer_id;
```

Result:

customer_id	customer_name	order_id	order_date	total_amount
1	John Doe	101	2025-01-01	250.00
1	John Doe	103	2025-01-03	150.00
2	Jane Smith	102	2025-01-02	300.00
NULL	NULL	104	2025-01-04	200.00

4. FULL OUTER JOIN

The FULL OUTER JOIN retrieves all rows from both tables. If there is no match, NULL values are returned for the columns of the non-matching table. MySQL does not directly support FULL OUTER JOIN, but you can achieve it using a UNION.

```
SELECT
    c.customer_id,
    c.name AS customer_name,
    o.order_id,
    o.order_date,
    o.total_amount
FROM
    customers c
LEFT JOIN
    orders o ON c.customer_id = o.customer_id
UNION
SELECT
    c.customer_id,
    c.name AS customer_name,
    o.order_id,
    o.order_date,
    o.total_amount
FROM
    customers c
RIGHT JOIN
    orders o ON c.customer_id = o.customer_id;
```

Result:

customer_id	customer_name	order_id	order_date	total_amount
1	John Doe	101	2025-01-01	250.00
1	John Doe	103	2025-01-03	150.00
2	Jane Smith	102	2025-01-02	300.00
3	Mike Johnson	NULL	NULL	NULL
NULL	NULL	104	2025-01-04	200.00