MySQL CHEAT SHEET

SQL

SQL stands for Structured Query Language. It allows you to access and manipulate databases.
SQL statements are used to perform tasks such as updating data on a database or retrieving data from a database. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

SQL Commands

- **CREATE**
  Used to create a new table in the database.

  ```
  CREATE TABLE table_name(
     column1 datatype,
      column2 datatype,...
  );
  ```

- **ALTER**
  Used to alter the contents of a table by adding some new column or attribute, or changing some existing attributes.

  -
    ```
    ALTER TABLE table_name
    ADD column_name datatype;
    ```

  -
    ```
    ALTER TABLE table_name
    MODIFY column_name
    datatype;
    ```

- **DROP**
  Used to delete the structure and record stored in the table.

  ```
  DROP TABLE table_name;
  ```

- **TRUNCATE**
  Used to delete all the rows from the table, and free up the space in the table.

  ```
  TRUNCATE TABLE table_name;
  ```

- **INSERT**
  Used to insert data in the row of a table.

  ```
  INSERT INTO table_name (column1,
  column2, column3,...) VALUES
  (value1, value2, value3, ...);
  ```

- **UPDATE**
  Used to update the value of a table's column.

  ```
  UPDATE table_name
  SET column1 = value1, column2 =
  value2,...
  WHERE condition;
  ```

- **DELETE**
  Used to delete one or more rows in a table.

  ```
  DELETE FROM table_name WHERE
  condition;
  ```

MySQL CHEAT SHEET

DataTrained
Keep Skilling. Keep Growing

SAMPLE DATA

Country Table

Id	Name	Population	Area
1	India	1,380,004,385	2,973,190
2	USA	336,369,717	9,147,420
3	Russia	336,369,717	16,376,870
...	...	...	...

City Table

Id	Name	Population	Country_Id	Rating
1	Delhi	10,927,986	1	7
2	New York	8,175,133	2	5
3	Moscow	10,381,222	3	6
...	...	...	...	...

SINGLE TABLE QUERIES

➢ Fetch all columns from the country table:

```
SELECT *
FROM country;
```

➢ Fetch the id and name columns from the city table:

```
SELECT id, name
FROM city;
```

➢ Fetch city names sorted by the rating column in the default ASCending order:

```
SELECT name
FROM city
ORDER BY rating ASC;
```

➢ Fetch city names sorted by the rating column in the DESCending order:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

ALIASES

It is a temporary name that is given to a table or a column while writing a query.

- **Column**

```
SELECT name AS city_name
FROM city;
```

- **Table**

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
ON ci.country_id = co.id;
```

COMPARISON OPERATORS

➢ Fetch names of cities that have a rating above 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

➢ Fetch names of cities that are neither Berlin nor Madrid:

```
SELECT name
FROM city
WHERE name != 'Berlin'
AND name != 'Madrid';
```

TEXT OPERATORS

➢ Fetch names of cities that start with a 'P' or end with an 's':

```
SELECT name
FROM city
WHERE name LIKE 'P%'
OR name LIKE '%s';
```

➢ Fetch names of Countries that start with any letters followed by 'ia' (like India or Russia):

```
SELECT name
FROM country
WHERE name LIKE '_ _ _ia';
```

MySQL CHEAT SHEET

OTHER OPERATORS

➢ Fetch names of cities that have a population between 500K and 12M:

```
SELECT name
FROM city
WHERE population
BETWEEN 500000 AND 12000000;
```

➢ Fetch names of cities that don't miss a rating value:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

➢ Fetch names of cities that are in countries with IDs 1, 3, or 7

```
SELECT name
FROM city
WHERE country_id IN (1, 3, 7);
```

MULTIPLE TABLE QUERIES

- **JOIN**

JOIN (or INNER JOIN) returns rows that have matching values in both tables

```
SELECT city.name, country.name
FROM city
JOIN country
ON city.country_id = country.id
```

City Table	
Name	**Country_Id**
Delhi	1
New York	2
Moscow	3
...	...

Country Table	
Country_Id	**Name**
1	India
2	USA
3	Russia
...	...

- **LEFT JOIN**

It returns all rows from the left table with corresponding rows from the right table. If there's no matching row, NULLs are returned as values from the second table

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
ON city.country_id = country.id;
```

City Table	
Name	**Country_Id**
Delhi	1
New York	2
Moscow	3
...	...

Country Table	
Country_Id	**Name**
1	India
2	USA
NULL	NULLI
...	...

- **RIGHT JOIN**

It returns all rows from the right table with corresponding rows from the left table. If there's no matching row, NULLs are returned as values from the left table.

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
ON city.country_id = country.id;
```

City Table	
Name	**Country_Id**
Delhi	1
New York	2
NULL	NULL
...	...

Country Table	
Country_Id	**Name**
1	India
2	USA
3	Russia
...	...

- **FULL JOIN**
  Full Join (or Full Outer Join) returns all rows from both tables – if there is no matching row in the second table, NULLs are returned.

```
SELECT city.name, country.name
FROM city
FULL JOIN country
ON city.country_id = country.id;
```

City Table	
Name	Country_Id
Delhi	1
New York	2
NULL	NULL
...	...

Country Table	
Country_Id	Name
1	India
NULL	NULL
3	Russia
...	...

- **CROSS JOIN**
  It returns all possible combinations of rows from both tables. There are two syntaxes available.

```
SELECT city.name, country.name
FROM city CROSS JOIN country;
```

```
SELECT city.name, country.name
FROM city, country;
```

City Table	
Name	Country_Id
New York	2
New York	2
Delhi	1
Delhi	1

Country Table	
Country_Id	Name
1	India
2	USA
1	India
2	USA

GROUP BY

GROUP BY groups together rows that have the same values in specified columns.

It computes summaries (aggregates) for each unique combination of values.

```
SELECT Country_id, COUNT (City_Name)
AS Count FROM City Table
GROUP BY Country_Id;
```

City Table	
City_Name	Country_Id
Delhi	1
New York	2
Moscow	3
Mumbai	1
Dallas	2
Omsk	3
Bengaluru	1
Los Angeles	2
Petersburg	3

City Table	
Country_Id	Count
1	3
2	3
3	3

AGGREGATE FUNCTION

- **avg(expression):**
  average value for rows within the group.
- **count(expression):**
  count of values for rows within the group.
- **max(expression)**
  maximum value within the group.
- **min(expression):**
  minimum value within the group.
- **sum(expression):**
  sum of values within the group.

For Ex.

- Find out the number of cities:

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

- Find out the number of cities with non-null ratings:

```
SELECT COUNT(rating)
FROM city;
```

- Find out the number of distinctive country values:

```
SELECT COUNT (DISTINCT
country_id) FROM city;
```

- Find out the smallest and the greatest country populations:

```
SELECT
MIN(population),
MAX(population)
FROM country;
```

- Find out the total population of cities in respective countries:

```
SELECT country_id,
SUM(population)
FROM city
GROUP BY country_id;
```

- Find out the average rating for cities in respective countries if the average is above 3.0:

```
SELECT country_id,
AVG(rating) FROM city
GROUP BY country_id HAVING
AVG(rating) > 3.0;
```

SUBQUERIES

A subquery is a query that is nested inside another query, or inside another subquery.

There are different types of subqueries.

- **Single Value**
  The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators =, <, <=, >, or >=.

  This query finds cities with the same rating as Paris:

```
SELECT name FROM city
WHERE rating = (
SELECT rating
FROM city
WHERE name = 'Delhi'

);
```

- **Multiple Value**
  A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators IN, EXISTS, ALL, or ANY.

  This query finds cities in countries that have a population above 20M:

```
SELECT name
FROM city
WHERE country_id IN (
SELECT country_id
FROM country
WHERE population>20000000
);
```

Keys in MySQL

A key can be a single column or a group of columns used to uniquely identify the rows of a table. SQL keys are a means to ensure that no row will have duplicate values. They are also a means to establish relations between multiple tables in a database.

Types of Keys

- **Primary Key:**
  - They uniquely identify a row in a table.
  - Only a single primary key for a table.
  - The primary key column cannot have any NULL values.
  - The primary key must be unique for each row.

  Ex.

```
CREATE TABLE Student (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Class int, PRIMARY KEY (ID)
);
```

- **Foreign Key:**
  - Foreign keys are keys that reference the primary keys of some other table.
  - They establish a relationship between 2 tables and link them up.
  - Example:
    In the below example, a table called Orders is created with some given attributes, and its Primary Key is declared to be OrderID and Foreign Key is declared to be PersonId referenced from the Person's table. A person's table is assumed to be created beforehand.

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
PRIMARY KEY (OrderID),
FOREIGN KEY (PersonID)
REFERENCES
Persons(PersonID)
);
```

- **Super Key:**
  It is a group of single or multiple keys which identifies a row of a table.

- **Candidate Key:**
  It is a collection of unique attributes that can uniquely identify tuples in a table.

- **Compound Key:**
  It is a collection of more than one record that can be used to uniquely identify a specific record.

- **Composite Key:**
  Collection of more than one column that can uniquely identify rows in a table.

Important SQL Keywords

Keyword	Description	Example
ADD	Will add a new column to an existing table.	ALTER TABLE student ADD email_address VARCHAR(255)
ALTER TABLE	Adds edits or deletes columns in a table	ALTER TABLE student DROP COLUMN email_address;
ALTER COLUMN	Can change the datatype of a table's column	ALTER TABLE student ALTER COLUMN phone VARCHAR(15)
AS	Renames a table/column with an alias existing only for the query duration.	SELECT name AS student_name, phone FROM student;
ASC	Used in conjunction with ORDER BY to sort data in ascending order.	SELECT column1, column2, … FROM table_name ORDER BY column1, column2, … ASC;
DESC	Used in conjunction with ORDER BY to sort data in descending order.	SELECT column1, column2, … FROM table_name ORDER BY column1, column2, … DESC;
CHECK	Constrains the value which can be added to a column.	CREATE TABLE student(fullName varchar(255), age INT, CHECK(age >= 18));
CREATE DATABASE	Creates a new database.	CREATE DATABASE student;
DEFAULT	Sets the default value for a given column.	CREATE TABLE products(ID int, name varchar(255) DEFAULT 'Username', from date DEFAULT GETDATE());
DELETE	Delete values from a table.	DELETE FROM users WHERE user_id= 674;
DROP COLUMN	Deletes/Drops a column from a table.	ALTER TABLE student DROP COLUMN name;
DROP DATABASE	Completely deletes a database with all its content within.	DROP DATABASE student;
DROP DEFAULT	Removes a default value for a column.	ALTER TABLE student ALTER COLUMN age DROP DEFAULT;
DROP TABLE	Deletes a table from a database.	DROP TABLE students;
FROM	Determines which table to read or delete data from.	SELECT * FROM students;
IN	Used with WHERE clause for multiple OR conditionals.	SELECT * FROM students WHERE name IN('Scaler', 'Interviewbit', 'Academy');
ORDER BY	Used to sort given data in Ascending or	SELECT * FROM student ORDER BY age ASC

	Descending order.	
SELECT DISTINCT	Works in the same war as SELECT, except that only unique values are included in the results.	SELECT DISTINCT age from student;
TOP	Used in conjunction with SELECT to select a fixed number of records from a table.	SELECT TOP 5 * FROM students;
VALUES	Used along with the INSERT INTO keyword to add new values to a table.	INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');
WHERE	Filters given data based on some given condition.	SELECT * FROM students WHERE age >= 18;
UNIQUE	Ensures that all values in a column are different.	UNIQUE (ID)
UNION	Used to combine the result-set of two or more SELECT statements.	SELECT column_name(s) FROM Table1 UNION SELECT column_name(s) FROM Table2;
UNION ALL	Combines the result set of two or more SELECT statements(it allows duplicate values)	SELECT City FROM table1 UNION ALL SELECT City FROM table2 ORDER BY City;
SELECT TOP	Used to specify the number of records to return.	SELECT TOP 3 * FROM Students;
LIMIT	Puts a restriction on how many rows are returned from a query.	SELECT * FROM table1 LIMIT 3;
UPDATE	Modifies the existing records in a table.	UPDATE Customers SET ContactName = 'Scaler', City = 'India' WHERE CustomerID = 1;
SET	Used with UPDATE to specify which columns and values should be updated in a table.	UPDATE Customers SET ContactName = 'Scaler', City= 'India' WHERE CustomerID = 1;
IS NULL	Column values are tested for NULL values using this operator.	SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL;
LIKE	Used to search for a specified pattern in a column.	SELECT * FROM Students WHERE Name LIKE 'a%';
ROWNUM	Returns a number indicating the order in which Oracle selects the row from a table or set of joined rows.	SELECT * FROM Employees WHERE ROWNUM < 10;
GROUP BY	Groups rows that have the same values into summary rows.	SELECT COUNT(StudentID), State FROM Students GROUP BY State;
HAVING	Enables the user to specify conditions that filter which group results appear in the results.	HAVING COUNT(CustomerID) > 5;

SQL Operators

➢ **Arithmetic Operators:**
It allows the user to perform arithmetic operations in SQL.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

➢ **Bitwise operators:**
It is used to perform Bit manipulation operations in SQL.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

➢ **Relational Operators:**
It is used to perform relational expressions in SQL, i.e those expressions whose value either results in true or false.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

➢ **Logical Operators::**

It used to combine 2 or more relational statements into 1 compound statement whose truth value is evaluated as a whole.

Operator	Description
All	True if all subqueries meet the given condition.
AND	True if all the conditions turn out to be true.
ANY	True if any of the subqueries meet the given condition.
BETWEEN	True if the operand lies within the range of the conditions.
EXISTS	True if the subquery returns one or more records
IN	True if the operands to at least one of the operands in a given list of expressions.
LIKE	True if the operand and some given pattern match.
NOT	Displays some record if the set of given conditions is False
OR	True if any of the conditions turn out to be True
SOME	True if any of the Subqueries meet the given condition.