SQL NOTES

BY- NIDHI KUSHWAHA

Data

- Data is a collection of facts, such as numbers, words, measurements, observations or descriptions of things.

Examples-

- Numbers (e.g., 42,3.14)
- Words (e.g., “Hello”, “SQL”)
- Measurements (e.g., height , weight)
- Observations (e.g., “It is raining”)

Database

- A database is an organized collection of data, generally stored and accessed electronically from a computer system.
- It allow for efficient storage, retrieval, and management of data.

Examples of Databases:

- E-commerce platforms (e.g., Amazon)
- Social media platforms (e.g., Facebook)

Database Management System (DBMS)

- Database management system is a software which is used to manage the database.
- DBMS provides an interface to perform various operations like-

1. Creating databases, tables, and objects.
2. Inserting, updating, and deleting data.
3. Dropping databases, tables, and objects.
4. Provides data security.

- Some popular DBMS softwares are MS SQL SERVER,Oracle,MySQL,IBM,DB2,PostgreSQL etc.

Relational Database Management System(RDBMS)

- It is a type of DBMS that manage relational database.
- It helps to store, organize and retrieve data efficiently. RDBMS softwares are MySQL, PostgreSQL, SQL Server, and Oracle.

Table

- A table in a database is a collection of rows and columns.
- It is used to organize and store data in a structured format.

Row and Column

- A **row** is a horizontal arrangement of data moving from right to left. It is often referred to as a **record or a tuple**. It represents individual data entries.
- A **column** is a vertical arrangement of data moving from top to bottom. It is often referred to as an **attribute or a field**. It represents the characteristics or properties of the data.

Example**-** A "Customers" table might have columns like CustomerID**,** Name, Email**,** and Phone with each row representing a different customer.

SQL

- SQL stands for Structured Query Language was initially developed by IBM.
- Initially it was called as SEQUEL (Structure English Query Language)
- SQL is a programming language used to interact with database.
- SQL allows you to **create, read, update, and delete (CRUD)** data in a relational database.

SQL Data Types

- Data Types define the type of data that can be stored in a table column.
- It ensures data integrity and optimizes storage.

Commonly used Data Types are-

A. String Datatypes- It stores text strings.

1. **CHAR**-
   - Can store characters of fixed length.
   - The size parameter specifies the column length in characters can be from 0 to 255.

2. **VARCHAR**-
   - Can store characters up to given length.
   - The size parameter specifies the maximum column length in characters can be from 0 to 65535.

B. Numeric Datatypes- These datatypes store numbers, both integer and floating-point number.

1. **INT**-
- Used for storing whole numbers without decimal.

2. **FLOAT**-
- Used for storing numbers without decimal.
- Float gives approximate value while performing calculations.

3. **Decimal (P, S)**-
- Used to store numbers with a fixed number of decimal places.
- Decimal gives exact value.
- Total range of digits can be 1-38. (Precision-P)
- Range after decimal should be 0-30. (Scale-S)

C. **Date and Time Datatypes**- These datatypes stores date and time.

- **DATE**- Stores date values (Format:- YYYY-MM-DD).
- **TIME**- Stores time values (Format:- HH:MM:SS).
- **DATETIME**- Stores both date and time values (Format:- YYYY-MM-DD HH:MM:SS).

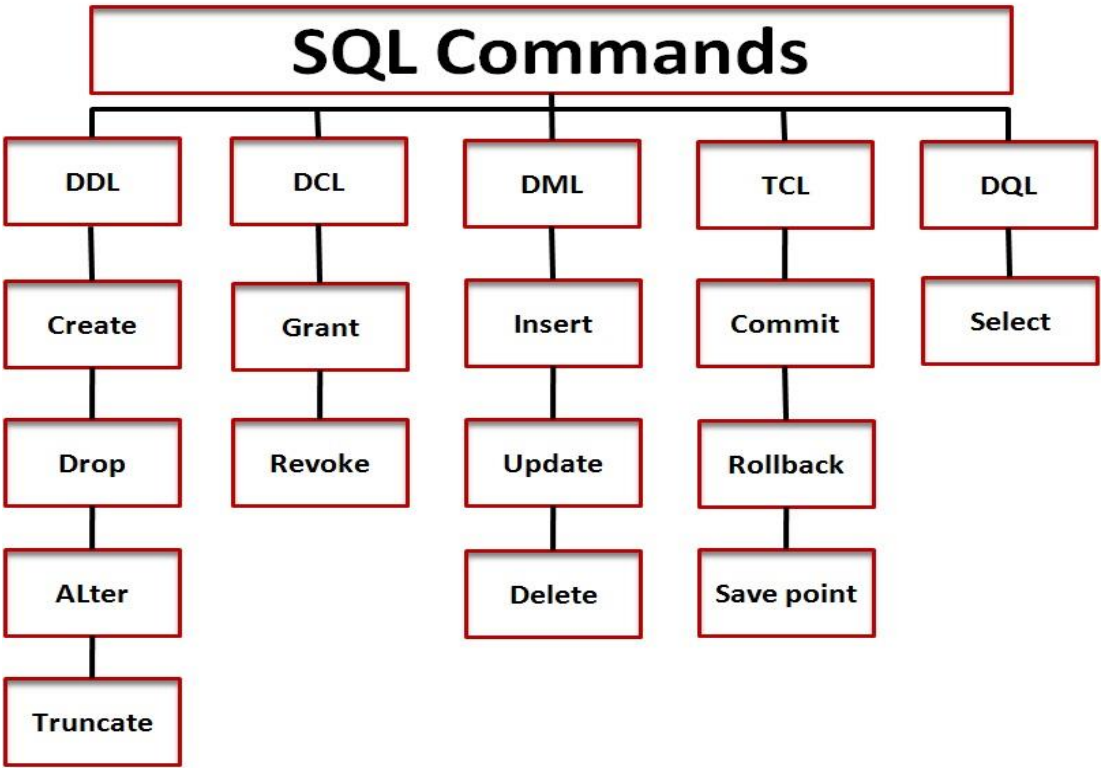
D. **BIT Datatype**- BIT data type can store either True or False values (represents by 1 and 0).

Example-

```
CREATE TABLE Employees (
   EmployeeID INT,
   FirstName VARCHAR(50),
   LastName CHAR(20),
   Salary DECIMAL(10, 2),
   BirthDate DATE,
   HireTime TIME,
   IsActive BIT
);
```

SQL Commands

- SQL commands are instructions.
- It is used to communicate with the database to perform tasks, functions and queries of data.
- There are five types of SQL commands.



1. **Data Definition Language (DDL)**- DDL commands are used to define and manage the structure or schema of a database.

2. **Data Manipulation Language(DML)**- DML commands are used to manipulate or manage the data stored in a database. These commands are specifically designed to modify the actual data stored in the database.

3. **Data Control Language(DCL)**- DCL commands are specifically designed for managing security and permissions within a database.

4. **Transaction Control Language(TCL)**- TCL commands manages transactions in a database.

5. **Data Query Language(DQL)**- DQL commands are used primarily for querying and retrieving data from a database.

Create Command

- Create command is used for creating a new database and a table.

1. Create a Database-

Syntax-

CREATE DATABASE database_name;

Example-

CREATE DATABASE SalesDB;

USE DATABASE- It is used to select a database from a list of database available in the system.

Syntax-

USE database_name;

Example-

USE SalesDB;

2. **Create Table-**

Syntax-

```
CREATE TABLE table_name(
 column1 datatype,
 column2 datatype,
 column3 datatype,
 .....
 columnN datatype,
);
```

Example-

```
CREATE TABLE CUSTOMERS(
 ID INT,
 NAME VARCHAR (20),
 AGE INT,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
 );
```

Drop Command

- DROP command is used to permanently remove the entire table or database, along with its structure and the data.
- DROP command cannot be rolled back. Once executed, the data and structure cannot be recovered unless there's a backup.

1. Drop a Database-

Syntax-

```
DROP DATABASE database_name;
```

Example-

```
DROP DATABASE SalesDB;
```

2. Drop a Table-

Syntax-

```
DROP TABLE table_name;
```

Example-

```
DROP TABLE CUSTOMERS;
```

Insert Command

- INSERT command is used to insert new records into a table.

1. Insert data into all columns-

Syntax-

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

Example-

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00);
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

NOTE- You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

Syntax-

```
INSERT INTO table_name VALUES (value1, value2, value3, ...)
```

Example-

```
INSERT INTO CUSTOMERS VALUES (3, 'Kaushik', 23, 'Kota', 2000.00);
INSERT INTO CUSTOMERS VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00);
INSERT INTO CUSTOMERS VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00);
INSERT INTO CUSTOMERS VALUES (6, 'Muffy', 24, 'Indore', 10000.00 );
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00

2. Insert data into specific columns-

Syntax-

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

Example-

INSERT INTO CUSTOMERS (ID, NAME, SALARY)
VALUES (7, 'Bharti', 12000.00);

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

Select Command

- The SELECT command is used to retrieve/fetch the data from a database.

1. Select all columns/fields-

To select all columns from a table, you can use the asterisk (*):

Syntax-

SELECT * FROM table_name;

Example-

SELECT * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

2. Select specific columns/fields-

Syntax-

SELECT column1, column2, ... columnN FROM table_name;

Example-
To fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

SELECT ID, NAME, SALARY FROM CUSTOMERS;

ID	NAME	SALARY
1	Ramesh	2000.00
2	Khilan	1500.00
3	Kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Muffy	10000.00
7	Bharti	12000.00

Where Clause

- WHERE clause is used to filter records based on specified/given conditions. It allows you to retrieve only those rows that meet certain criteria.

Syntax-

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example-

To fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name Hardik.

Note- It is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

ID	NAME	SALARY
5	Hardik	8500.00

Example-

To fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000.

```
SELECT *
FROM CUSTOMERS
WHERE SALARY > 2000;
```

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

SQL Operators

- SQL operators are used to perform operations on data within queries.
- SQL operators are symbols or keywords that perform actions like comparing values, doing math or combining conditions in queries to filter, manipulate and analyze data in databases.

Different types of SQL operators:

1. Arithmetic Operators-

- Used to perform mathematical operations on numeric values.

Common operators-

a) **Addition (+):** Adds two values.

Example- Let a=10 and b=5

SELECT 10 + 5;

(a+b will give the result as 15)

b) Subtraction (-): Subtract one value from another.

Example- Let a=10 and b=5

SELECT 10 - 5;

(a-b will give the result as 5)

c) Multiplication (x): Multiplies two values.

Example- Let a=10 and b=5

SELECT 10 * 5;

(a*b will give the result as 50)

d) Division (/): Divides one value by another.

Example- Let a=10 and b=5

SELECT 10 / 5;

(a/b will give the result as 2)

e) Modulus (%): Returns the remainder of division.

Example- Let a=10 and b=5

SELECT 10 % 5;

(a%b will give the result as 0)

2. Comparison Operators-

- Used to compare two values.

Common operators-

a) **Equal (=):** Checks if two values are equal, if yes then condition becomes true.

Example-

SELECT * FROM CUSTOMERS WHERE ID = 3;

ID	NAME	AGE	ADDRESS	SALARY
3	Kaushik	23	Kota	2000.00

b) **Not equal to (!= or <>):** Checks if two values are not equal, if yes then condition becomes true.

Example-

SELECT * FROM CUSTOMERS WHERE SALARY != 2000;

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

c) **Greater than (>):** Checks if one value is greater than another.

Example-

SELECT * FROM CUSTOMERS WHERE SALARY > 2000;

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

d) **Less than (<):** Checks if one value is less than another.

Example-

SELECT * FROM CUSTOMERS WHERE SALARY < 2000;

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00

e) **Greater than or Equal To (>=):** Checks if one value is greater than or equal to another.

Example-

SELECT * FROM CUSTOMERS WHERE SALARY >= 2000;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

f) **Less than or Equal To (<=):** Checks if one value is less than or equal to another.

Example-

SELECT * FROM CUSTOMERS WHERE SALARY <= 2000;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00

3. Logical Operators-

- Used to combine multiple conditions.

Common Operators-

a) **AND:** Returns true if all conditions are true.

Syntax-

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2;
```

Example-

To fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

ID	NAME	SALARY
6	Muffy	10000.00

b) **OR:** Returns true if any condition is true.

Syntax-

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2;
```

Example-

To fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

ID	NAME	SALARY
3	Kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Muffy	10000.00
7	Bharti	12000.00

c) **NOT:** It is used to negate a condition. This operator is used to exclude certain records means it helps you find records where the condition is not true.

Syntax-

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example-

To fetch the ID, Name and Salary fields from the CUSTOMERS table, whose salary is not 2000.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NOT SALARY  = 2000;
```

ID	NAME	SALARY
2	Khilan	1500.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Muffy	10000.00
7	Bharti	12000.00

IN Operator

- IN operator checks whether a value matches any value in a given list.
- The IN operator works similarly to using multiple OR conditions.

Syntax-

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Example-

Retrieve the ID, name, and address of customers living in either Mumbai, Delhi, or Bhopal.

```
SELECT ID, NAME, ADDRESS
FROM CUSTOMERS
WHERE ADDRESS IN('DELHI','BHOPAL','MUMBAI');
```

ID	NAME	ADDRESS
2	Khilan	Delhi
4	Chaitali	Mumbai
5	Hardik	Bhopal

BETWEEN Operator

- BETWEEN operator checks whether a value is within a range of values.
- It can be used with various data types such as numbers, dates, and strings.
- BETWEEN operator in SQL is similar to using the AND operator.

Syntax-

SELECT column1, column2, ...

FROM table_name

WHERE column_name BETWEEN low_value AND high_value;

Example-

Retrieve ID, NAME, AGE FROM CUSTOMERS WHERE AGE BETWEEN 24 AND 27 from the CUSTOMERS table.

SELECT ID, NAME, AGE

FROM CUSTOMERS

WHERE AGE BETWEEN 24 AND 27;

ID	NAME	AGE
2	Khilan	25
4	Chaitali	25
5	Hardik	27
6	Muffy	24

Example-

Retrieve the details from CUSTOMERS table WHERE SALARY BETWEEN 2000 AND 10000.

```
SELECT *
FROM CUSTOMERS
WHERE SALARY BETWEEN 2000 AND 10000;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00

NULL

- NULL values represent missing value or unknown data.
- NULL is neither a space or zero(0).
- NULL can’t be compared with any value/anything, not even to another NULL.

Example-

If you have a CUSTOMERS table and you want to insert a record for Bharti without providing her age and address:

INSERT INTO CUSTOMERS (ID, NAME, SALARY)

VALUES (7, 'Bharti', 12000.00);

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

In this case, if the AGE AND ADDRESS column is set to default to NULL, BHARTI AGE AND ADDRESS value will be NULL.

Example-

If you do not want to provide a value for a column, you can assign NULL to that column.

In this statement, we are inserting a new record into the CUSTOMERS table.

The ID is set to 8, the NAME is set to 'Rahul', AGE is set to 25, ADDRESS is set to 'Delhi' and the SALARY is set to NULL.

```
INSERT INTO CUSTOMERS (ID, NAME,AGE,ADDRESS,SALARY)
VALUES (8, 'Rahul',25,'Delhi', NULL);
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL

IS NULL

- This condition is used to check whether a column contains a NULL value.

Syntax-

SELECT * FROM table_name WHERE column_name IS NULL;

Example-

Suppose you have a CUSTOMERS table, and you want to find all customers whose salary is not provided (i.e., NULL):

SELECT * FROM CUSTOMERS WHERE SALARY IS NULL;

ID	NAME	AGE	ADDRESS	SALARY
8	Rahul	25	Delhi	NULL

IS NOT NULL

This condition is used to check whether a column does not contain a NULL value.

Syntax:

SELECT * FROM table_name WHERE column_name IS NOT NULL;

Example-

Using the same CUSTOMERS table, if you want to find all customers who have a salary specified (i.e., not NULL):

SELECT * FROM CUSTOMERS WHERE SALARY IS NOT NULL;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00

DISTINCT

- DISTINCT keyword is used to return only unique values in a result set by removing duplicate rows from the query results.

Syntax-

SELECT DISTINCT column1, column2, ...

FROM table_name;

Example-

Suppose you have a table called Customers with the following data:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

If you want to retrieve distinct address from this table, you can use the DISTINCT keyword like this:

SELECT DISTINCT ADDRESS

FROM CUSTOMERS;

ADDRESS
NULL
Ahmedabad
Bhopal
Delhi
Indore
Kota
Mumbai

If you want to select all unique rows from a table, you can use:

Syntax-

SELECT DISTINCT *

FROM table_name;

Note- This will return all columns from the table but only show distinct rows. However, keep in mind that using DISTINCT * may not be efficient if your table has many columns. It's usually better to specify only the columns you need.

Alias

- In SQL, an alias is a temporary name given to a table or column, mainly used to make queries more readable and concise.
- Aliases are temporary. They only exist for the duration of the query in which they are used.
- Aliases are created using the AS keyword, but using AS is optional.

Syntax-

The basic syntax of a **table** alias is as follows-

SELECT column_name AS alias_name

FROM table_name

WHERE [condition];

The basic syntax of a **column** alias is as follows.

SELECT column_name AS alias_name

FROM table_name

WHERE [condition];

Example-

Let's say we want to retrieve data from both the CUSTOMERS and ORDERS tables so we'll use aliases for better readability.

Table 1 − CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

Table 2 − ORDERS Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
100	2009-10-08 00:00:00.000	3	3000.00
101	2009-10-08 00:00:00.000	3	1500.00
102	2009-11-20 00:00:00.000	2	1560.00
103	2008-05-20 00:00:00.000	4	2060.00

Example Query with Table Aliases:

We will assign aliases to the tables CUSTOMERS and ORDERS to make the query shorter and easier to read.

```
SELECT C.NAME, O.AMOUNT ,O.DATE
FROM CUSTOMERS AS C, ORDERS AS O
WHERE C.ID = O.CUSTOMER_ID;
```

Here,

- **C is the alias** for the **CUSTOMERS** table.
- **O is the alias** for the **ORDERS** table.
- We are selecting the NAME column from the CUSTOMERS table, and the AMOUNT and DATE columns from the ORDERS table.
- The JOIN condition is **C.ID = O.CUSTOMER_ID**, which connects the two tables based on the ID in the CUSTOMERS table and the CUSTOMER_ID in the ORDERS table.

Result-

NAME	AMOUNT	DATE
Khilan	1560.00	2009-11-20 00:00:00.000
Kaushik	3000.00	2009-10-08 00:00:00.000
Kaushik	1500.00	2009-10-08 00:00:00.000
Chaitali	2060.00	2008-05-20 00:00:00.000

Example Query with Column Aliases:

```
SELECT C.NAME AS CUSTOMER_NAME , O.AMOUNT AS ORDER_AMOUNT,
O.DATE AS ORDER_DATE
FROM CUSTOMERS AS C, ORDERS AS O
WHERE C.ID = O.CUSTOMER_ID;
```

Here,

- Customer_Name is the alias for the NAME column from the CUSTOMERS table.
- Order_Amount is the alias for the AMOUNT column from the ORDERS table.
- Order_Date is the alias for the DATE column from the ORDERS table.

Result-

CUSTOMER_NAME	ORDER_AMOUNT	ORDER_DATE
Khilan	1560.00	2009-11-20 00:00:00.000
Kaushik	3000.00	2009-10-08 00:00:00.000
Kaushik	1500.00	2009-10-08 00:00:00.000
Chaitali	2060.00	2008-05-20 00:00:00.000

In SQL Server, the AS keyword is commonly used to create aliases, it is not mandatory. You can create aliases without using AS by simply specifying the alias name immediately after the column or table name.

Here are the ways to create aliases without the AS keyword:

1. **Column Alias**: Just place the alias name after the column name.

   Syntax-
   SELECT column_name alias_name;

2. **Table Alias**: Just place the alias name after the table name.

   Syntax-
   SELECT column_name FROM table_name alias_name;

   Example- Here's a full query using both types of aliases without the AS keyword.

   ```
   SELECT C.NAME CUSTOMER_NAME , O.AMOUNT ORDER_AMOUNT
   FROM CUSTOMERS C, ORDERS O
   WHERE C.ID = O.CUSTOMER_ID;
   ```

CUSTOMER_NAME	ORDER_AMOUNT
Khilan	1560.00
Kaushik	3000.00
Kaushik	1500.00
Chaitali	2060.00

Create a clone table/duplicate table/backup table

In SQL Server, both **SELECT INTO** and **INSERT INTO SELECT** can be used to copy data from one table to another, but they are used differently.

1. SELECT INTO-

- SELECT INTO is used to create a new table and copy data into it at the same time.
- It duplicates both the structure and data from an existing table into a new one.

A) Copy all Columns into a New Table-

Syntax-

```
SELECT * INTO new_table
FROM existing_table
WHERE condition(Optional);
```

Example-

```
SELECT * INTO BACKUP_CUSTOMERS
FROM CUSTOMERS
WHERE ADDRESS = 'Delhi';

SELECT * FROM BACKUP_CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

This will create a new table backup_customers and copy all rows from the customers table where address= 'Delhi'.

B) Copy only Selected or Few Columns into a New Table-

Syntax-

```
SELECT columns
INTO new_table
FROM existing_table
WHERE condition(Optional);
```

Example-

```
SELECT NAME, ADDRESS INTO BACKUP_CUSTOMERSDETAIL
FROM CUSTOMERS;

SELECT * FROM BACKUP_CUSTOMERSDETAIL;
```

NAME	ADDRESS
Ramesh	Ahmedabad
Khilan	Delhi
Kaushik	Kota
Chaitali	Mumbai
Hardik	Bhopal
Muffy	Indore
Bharti	NULL
Rahul	Delhi
Muffy	Delhi

2. INSERT INTO SELECT-

- INSERT INTO SELECT is used to copy data or transfer data from one table to another table that already exists in your database.
- The target table must have a structure (columns) that matches or is compatible with the source table (the table from which you're copying data).

A) Copy only Selected or Few Columns from one Table to Another Table-

Syntax-

```
INSERT INTO target_table (column1, column2, ...)
SELECT column1, column2, ...
FROM source_table
WHERE condition;
```

Example-

INSERT INTO SELECT statement to copy data from CUSTOMERS to BACKUP_CUSTOMERS for customers who are older than 30.

```
INSERT INTO BACKUP_CUSTOMERS(ID, NAME, AGE, ADDRESS, SALARY)
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE AGE > 30;

SELECT * FROM BACKUP_CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00
1	Ramesh	32	Ahmedabad	2000.00

Here,

- **INSERT INTO BACKUP_CUSTOMERS:** This specifies that you are inserting data into the BACKUP_CUSTOMERS table.
- **(ID, NAME, AGE, ADDRESS, SALARY):** These are the columns in the BACKUP_CUSTOMERS table that will receive the data.
- **SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS:** This selects the corresponding columns from the CUSTOMERS table.
- **WHERE AGE > 30:** This condition ensures that only rows where the AGE is greater than 30 are copied from CUSTOMERS to BACKUP_CUSTOMERS.

B) Copy All Columns from one Table to Another Table-

Syntax-

```
INSERT INTO target_table
SELECT *
FROM source_table
WHERE condition;
```

Example-

```
INSERT INTO BACKUP_CUSTOMERS
SELECT *
FROM CUSTOMERS
WHERE ADDRESS ='Delhi';

SELECT * FROM BACKUP_CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

Delete Command

- The DELETE command is used to remove specific rows from a table based on a condition specified in the WHERE clause, or it can remove all rows if no condition is given, while keeping the table structure and any remaining data intact.
- DELETE can be rolled back if used within a transaction.
- It is slower than TRUNCATE because it logs each row deletion individually.

1. Deleting Specific Rows:

Syntax-

DELETE FROM table_name WHERE condition;

Example- If you want to delete record from the customers table where id =1.

DELETE FROM CUSTOMERS WHERE ID = 1;

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

2. Deleting All Rows:

Syntax-

DELETE FROM table_name;

Example-

If you want to delete all rows from a table without dropping the table structure.

DELETE FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY

Truncate Command

- The TRUNCATE command removes all rows from a table and retains the table structure for future use. Unlike DELETE, you cannot specify conditions—it clears the entire table in one go.
- It cannot be rolled back in some databases.
- Faster than DELETE because it doesn't log individual row deletions.

Syntax-

TRUNCATE TABLE table_name;

Example-

TRUNCATE TABLE CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY

Update Command

- Update command is used to modify/update the existing records in a table.

Syntax-

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Example-

If we want to update the ADDRESS for a customer whose ID number is 6 in the Customers table, the query would be:

UPDATE CUSTOMERS

SET ADDRESS = 'Pune'

WHERE ID = 6;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Pune	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

Example-

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be –

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1000.00
2	Khilan	25	Pune	1000.00
3	Kaushik	23	Pune	1000.00
4	Chaitali	25	Pune	1000.00
5	Hardik	27	Pune	1000.00
6	Muffy	24	Pune	1000.00
7	Bharti	NULL	Pune	1000.00
8	Rahul	25	Pune	1000.00
9	Muffy	24	Pune	1000.00

Alter Command

- The ALTER command is used for adding, deleting, or modifying the structure of an existing table.

Common Operations with ALTER:

1. **Adding Columns:** You can use the ALTER statement to add new columns to an existing table.

Syntax-

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

Example-

ALTER TABLE ORDERS

ADD Address VARCHAR(255);

OID	DATE	CUSTOMER_ID	AMOUNT	Address
100	2009-10-08 00:00:00.000	3	3000.00	NULL
101	2009-10-08 00:00:00.000	3	1500.00	NULL
102	2009-11-20 00:00:00.000	2	1560.00	NULL
103	2008-05-20 00:00:00.000	4	2060.00	NULL

2. **Modifying Columns:** You can change the data type or properties of existing columns.

Syntax-

ALTER TABLE table_name

ALTER COLUMN column_name new_datatype;

Example-

ALTER TABLE ORDERS

ALTER COLUMN AMOUNT DECIMAL(12, 1);

OID	DATE	CUSTOMER_ID	AMOUNT	Address
100	2009-10-08 00:00:00.000	3	3000.0	NULL
101	2009-10-08 00:00:00.000	3	1500.0	NULL
102	2009-11-20 00:00:00.000	2	1560.0	NULL
103	2008-05-20 00:00:00.000	4	2060.0	NULL

3. **Deleting Columns:** You can remove unwanted columns from a table using ALTER.

Syntax-

ALTER TABLE table_name

DROP COLUMN column_name;

Example-

ALTER TABLE ORDERS

DROP COLUMN ADDRESS;

OID	DATE	CUSTOMER_ID	AMOUNT
100	2009-10-08 00:00:00.000	3	3000.0
101	2009-10-08 00:00:00.000	3	1500.0
102	2009-11-20 00:00:00.000	2	1560.0
103	2008-05-20 00:00:00.000	4	2060.0

RENAME

- In SQL Server, the RENAME operation can be used to rename/change the names of database objects such as tables, columns, indexes, etc.
- You can rename a table or column within a table using the **sp_rename stored procedure.**
- **sp_rename** is a system stored procedure which helps to rename tables, columns, indexes, etc. in sql server.

Here are common scenarios for renaming:

1. ## Renaming a Table-

Syntax-

sp_rename 'old_table_name', 'new_table_name';

Example-

sp_rename 'ORDERS ', 'CUSTOMER_ORDERS';

2. ### Renaming a Column-

Syntax-

sp_rename 'table_name.old_column_name', 'new_column_name';

Example-

sp_rename 'ORDERS.DATE_TIME', 'DATE';

Order By Clause

The ORDER BY clause in SQL is used to sort the result set/data in either ascending or descending order, based on one or more columns.

Syntax-

SELECT column1, column2, ...

FROM table_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

Explanation:

- **column1, column2, ...:** Columns that you want to sort the data by.
- **ASC:** Sorts the data in ascending order (smallest to largest). This is the default.
- **DESC:** Sorts the data in descending order (largest to smallest).

1. Sorting in Ascending Order (Default)

Example- Retrieves all Customers and sorts them by their name in ascending order.

```
SELECT * FROM CUSTOMERS

ORDER BY NAME;
```

ID	NAME	AGE	ADDRESS	SALARY
7	Bharti	NULL	NULL	12000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	Kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Muffy	24	Indore	10000.00
9	Muffy	24	Delhi	10000.00
8	Rahul	25	Delhi	NULL
1	Ramesh	32	Ahmedabad	2000.00

2. Sorting in Descending Order

Example- Retrieves all Customers and sorts them by their name in descending order.

```
SELECT * FROM CUSTOMERS

ORDER BY NAME DESC;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00
6	Muffy	24	Indore	10000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
4	Chaitali	25	Mumbai	6500.00
7	Bharti	NULL	NULL	12000.00

3. Sorting by Multiple Columns

Example- Retrieves all Customers and sorts them by their name in Ascending order and address in Descending order.

```
SELECT * FROM CUSTOMERS

ORDER BY NAME,ADDRESS DESC;
```

ID	NAME	AGE	ADDRESS	SALARY
7	Bharti	NULL	NULL	12000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	Kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Muffy	24	Indore	10000.00
9	Muffy	24	Delhi	10000.00
8	Rahul	25	Delhi	NULL
1	Ramesh	32	Ahmedabad	2000.00

Explanation:

NAME: The result is first sorted alphabetically by the NAME column.

ADDRESS: If two rows have the same NAME, they are sorted by ADDRESS in descending order.

Group By Clause

- The GROUP BY clause is used to group rows that have the same values in specified columns.
- It is used for organizing similar data into groups.
- It's often used with aggregate functions such as COUNT (), SUM (), AVG (), MAX (), and MIN () to perform calculations on each group of rows.

Syntax-

SELECT column1, aggregate_function(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2, ...,

ORDER BY column1, column2, …;

Example- Write an SQL query to count how many times each customer appears in the CUSTOMERS table.

SELECT NAME, COUNT(ID) AS CUSTOMERS

FROM CUSTOMERS

GROUP BY NAME;

NAME	CUSTOMERS
Bharti	1
Chaitali	1
Hardik	1
Kaushik	1
Khilan	1
Muffy	2
Rahul	1
Ramesh	1

Example- If you want to know the total amount of the salary on each customer.

```
SELECT NAME, SUM(SALARY) AS TotalSalary
FROM CUSTOMERS
GROUP BY NAME;
```

NAME	TotalSalary
Bharti	12000.00
Chaitali	6500.00
Hardik	8500.00
Kaushik	2000.00
Khilan	1500.00
Muffy	20000.00
Rahul	NULL
Ramesh	2000.00

Example- Write an SQL query to calculate the total salary for each customer in the CUSTOMERS table whose salary is greater than 3000.

```
SELECT NAME, SUM(SALARY) AS TotalSalary
FROM CUSTOMERS
WHERE SALARY > 3000
GROUP BY NAME;
```

NAME	TotalSalary
Bharti	12000.00
Chaitali	6500.00
Hardik	8500.00
Muffy	20000.00

Having Clause

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.
- The HAVING clause in SQL is used to filter records based on aggregate functions (SUM(), COUNT(), MAX(), MIN(), or AVG()) after the GROUP BY clause is applied.

Syntax-

```
SELECT column1, aggregate_function(column2)
FROM table
WHERE condition
GROUP BY column1
HAVING aggregate_function(column2) condition
ORDER BY column1 ASC|DESC;
```

Example-

Let's say you want to find all employees whose SALARY is greater than 3000, and you also want to order the results by ID in ascending order.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
GROUP BY ID, NAME, SALARY
HAVING SALARY > 3000
ORDER BY ID ASC;
```

ID	NAME	SALARY
4	Chaitali	6500.00
5	Hardik	8500.00
6	Muffy	10000.00
7	Bharti	12000.00
9	Muffy	10000.00

Explanation:

- GROUP BY ID, NAME, SALARY: Groups the employees by their ID, NAME, and SALARY (since we want to work with each employee individually here).
- HAVING SALARY > 3000: Filters out employees whose salary is greater than 3000.
- ORDER BY ID ASC: Orders the result set by ID in ascending order.

Example-

Using the CUSTOMERS table below, write an SQL query to find the total salary for each address. Only include those addresses where the total salary exceeds 3000.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Muffy	24	Indore	10000.00
7	Bharti	NULL	NULL	12000.00
8	Rahul	25	Delhi	NULL
9	Muffy	24	Delhi	10000.00

```
SELECT ADDRESS, SUM(SALARY) AS SUM_SALARY
FROM CUSTOMERS
GROUP BY ADDRESS
HAVING AVG(SALARY) > 3000;
```

ADDRESS	SUM_SALARY
NULL	12000.00
Bhopal	8500.00
Delhi	11500.00
Indore	10000.00
Mumbai	6500.00

Explanation:

- SELECT ADDRESS, SUM(SALARY): This part selects the ADDRESS and the total (SUM) of SALARY for each address.
- FROM CUSTOMERS: Specifies the table from which to retrieve data.
- GROUP BY ADDRESS: Groups the results by each unique ADDRESS.
- HAVING SUM(SALARY) > 3000: Filters the grouped results to include only those groups where the total salary exceeds 3000.

TOP

- TOP command is used to specify the number of records to return.

Note – All the databases do not support the TOP clause. For example, MySQL supports the LIMIT clause to fetch limited number of records while Oracle uses the ROWNUM command to fetch a limited number of records.

Syntax-

SELECT TOP n column_name(s)

FROM table_name

WHERE condition;

Here, **n** is the **number of rows** you want to retrieve.

Example- Fetch the top 3 records from the CUSTOMERS table.

SELECT TOP 3 * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00

Constraints

- Constraints are rules applied to table column to enforce data integrity.
- It ensures the accuracy and reliability of the data in a database.
- They restrict the type of data that can be inserted into a table.
- Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

SQL supports different type of constraints they are-

1. **NOT NULL Constraint-**

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.

Syntax-

```
CREATE TABLE table_name (
   column_name datatype NOT NULL, ...
);
```

Example-

Let, create a new table called CUSTOMERS and adds five columns, three of which are ID, NAME and AGE, In this we specify not to accept NULLs −

```
CREATE TABLE CUSTOMERS (
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2)
);
```

FOR EXISTING TABLE-

Syntax-

ALTER TABLE table_name

ALTER COLUMN column_name datatype NOT NULL;

Example-

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the ADDRESS column, the query would be-

ALTER TABLE CUSTOMERS

ALTER COLUMN ADDRESS CHAR (25) NOT NULL;

2. DEFAULT CONSTRAINT-

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Syntax-

CREATE TABLE table_name (

column_name datatype DEFAULT default_value,

column_name datatype,

...

);

Example-

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS (
 ID INT,
 NAME VARCHAR (20),
 AGE INT,
 ADDRESS CHAR (25),
 SALARY DECIMAL (18, 2) DEFAULT 5000.00
);
```

FOR EXISTING TABLE-

Syntax-

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name DEFAULT default_value FOR column_name;
```

Example-

If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query-

```
ALTER TABLE CUSTOMERS
ADD CONSTRAINT DF_SAL DEFAULT 5000.00 FOR SALARY;
```

Explanation:

- DF_SAL: The name of the default constraint (make sure there's no space in constraint names).
- DEFAULT 5000.00: The default value for the SALARY column.
- FOR SALARY: Specifies the column where the default value will be applied.

Dropping a DEFAULT Constraint-

Syntax-

ALTER TABLE table_name

DROP CONSTRAINT constraint_name;

Example-

ALTER TABLE CUSTOMERS

DROP CONSTRAINT DF_SAL;

3. UNIQUE CONSTRAINT-

- A **UNIQUE** constraint ensures that all values in a column (or a group of columns) are unique. This means that no two rows can have the same value(s) in that column or set of columns.

Syntax-

```
CREATE TABLE table_name (
   column_name datatype UNIQUE,
   column_name datatype,
   ...
);
```

Example-

Let create a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS (
 ID INT,
 NAME VARCHAR (20),
 AGE INT UNIQUE,
 ADDRESS CHAR (25),
 SALARY DECIMAL (18, 2),
);
```

FOR EXISTING TABLE-

Syntax-

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (column_name);
```

Example-

If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column.

```
ALTER TABLE CUSTOMERS
ADD CONSTRAINT UQ_AGE UNIQUE(AGE);
```

Dropping a UNIQUE Constraint:

Syntax-

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

Example-

```
ALTER TABLE CUSTOMERS
 DROP CONSTRAINT UQ_AGE;
```

4. CHECK CONSTRAINT-

1. The CHECK constraint ensures that values entered into a column meet a specific condition:
   - **If the condition is TRUE**: The record is allowed and inserted into the table.
   - **If the condition is FALSE**: The record violates the constraint and isn't inserted.

2. CHECK constraint can be applied to a single or multiple columns.

Syntax-

```
CREATE TABLE table_name (
   column_name datatype,
   CONSTRAINT constraint_name CHECK (condition)
);
```

Example-

Imagine you have a CUSTOMERS table with a column for age, and you want to make sure that only customer who are 18 or older can be added to the table. You can use a CHECK constraint to create this rule.

```
CREATE TABLE CUSTOMERS (
 ID INT,
 NAME VARCHAR (20),
 AGE INT
 ADDRESS CHAR (25),
 SALARY DECIMAL (18, 2),
 CONSTRAINT chk_Age CHECK (AGE >= 18)
);
```

This means: "Only allow ages of 18 or more in the Age column."

If someone tries to enter an age below 18, the database will stop it and show an error.

FOR EXISTING TABLE-

Syntax-

ALTER TABLE table_name

ADD CONSTRAINT constraint_name CHECK (condition);

Example-

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column.

ALTER TABLE CUSTOMERS

ADD CONSTRAINT chk_Age CHECK (Age >= 18);

Dropping a CHECK Constraint:

Syntax-

ALTER TABLE table_name

DROP CONSTRAINT constraint_name;

Example-

ALTER TABLE CUSTOMERS

DROP CONSTRAINT chk_Age;

5. PRIMARY KEY-

- A Primary Key is a column (or a set of columns) that uniquely identifies each row/record in a table.
- Primary key must contain unique values, meaning no duplicate values are allowed, and it cannot contain NULL values.
- A table can have only one primary key, which may consist of a single column or multiple columns.
- It does not establish a relationship with another table.
- When multiple columns/fields are used as a primary key, they are called a composite key.
- It ensures entity integrity (each record is unique).

A. Single column Primary Key

The first CREATE table syntax is without using a constraint name and the second syntax is with a constraint name. Both are valid. It is good practice to use a constraint name for easy identification and management.

Syntax 1- Using Primary Key in the Column Definition

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

Example-

In this example, the id column is defined as the primary key directly within the column definition.

```
CREATE TABLE customer (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   salary DECIMAL(10, 2),
   address VARCHAR(255)
);
```

Syntax 2- Using a Separate CONSTRAINT for the Primary Key

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   ...
   CONSTRAINT PK_table_name PRIMARY KEY (column1)
);
```

Example-

In this example, the primary key is defined separately using a CONSTRAINT clause named pk_customer.

```
CREATE TABLE customer (
   id INT,
   name VARCHAR(100),
   age INT,
   salary DECIMAL(10, 2),
   address VARCHAR(255),
   CONSTRAINT pk_customer PRIMARY KEY (id)
);
```

B. Multiple column Primary Key (Composite Key)

The first CREATE table syntax is without using a constraint name and the second syntax is with a constraint name. Both are valid. It is good practice to use a constraint name for easy identification and management.

Syntax 1-

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

Example-

```
CREATE TABLE customer (
   id INT,
   name VARCHAR(100),
   age INT,
   salary DECIMAL(10, 2),
   address VARCHAR(255),
   PRIMARY KEY (id, name)
);
```

Syntax 2-

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   ...
   CONSTRAINT PK_table_name PRIMARY KEY (column1,column2)
);
```

Example-

```
CREATE TABLE customer (
   id INT,
   name VARCHAR(100),
   age INT,
   salary DECIMAL(10, 2),
   address VARCHAR(255),
   CONSTRAINT pk_customer PRIMARY KEY (id, name)
);
```

C. Alter Table

Syntax 1- Adding a Primary Key to a Single Column

```
ALTER TABLE table_name
ADD PRIMARY KEY (column_name);
```

Example-

Assuming the id column should be the primary key:

```
ALTER TABLE customer
ADD PRIMARY KEY (id);
```

Syntax 2- Adding a Composite Primary Key

ALTER TABLE table_name

ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2);

Example-

If you want to create a composite primary key using the id and name columns, you would use:

ALTER TABLE customer

ADD CONSTRAINT pk_customer PRIMARY KEY (id, name);

D. Drop the Primary Key Constraint

Syntax-

ALTER TABLE tablename

DROP CONSTRAINT constraint_name;

Example-

ALTER TABLE customer

DROP CONSTRAINT pk_customer;

6. FOREIGN KEY-

- A Foreign Key is a column or set of columns in table that refers to the primary key of another table.
- Foreign key can contain duplicate and NULL values.
- A table can have multiple foreign keys.
- It establishes a relationship/link between two or more tables.
- The table that contains the foreign key is called the child table and the table with the primary key is called the parent table or referenced table.
- It ensures referential integrity (ensures consistency between related tables).

1. CREATE TABLE-

Syntax 1- Creates a foreign key is without using a constraint name

```
CREATE TABLE child_table_name (
   child_col1 DATATYPE PRIMARY KEY,
   child_col2 DATATYPE,
   child_col3 DATATYPE,
   ...
   FOREIGN KEY (child_col11, child_col2)
   REFERENCES parent_table_name (parent_col1, parent_col2)
);
```

Example-

```
-- Creating the CUSTOMER table
CREATE TABLE CUSTOMERS (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   salary DECIMAL(10, 2),
   address VARCHAR(255)
);


-- Creating the ORDERS table with a foreign key reference to CUSTOMERS
CREATE TABLE ORDERS (
   ID INT NOT NULL,
   DATE DATETIME,
   CUSTOMER_ID INT,
   AMOUNT DECIMAL(10, 2),
      FOREIGN KEY (CUSTOMER_ID)
      REFERENCES CUSTOMERS (id)
);
```

Syntax 2- Creates a foreign key using a constraint name

```
CREATE TABLE child_table_name (
   child_col1 DATATYPE PRIMARY KEY,
   child_col2 DATATYPE,
   child_col3 DATATYPE,
   ...
   CONSTRAINT constraint_name
   FOREIGN KEY (child_col11, child_col2)
   REFERENCES parent_table_name (parent_col1, parent_col2)
);
```

Example-

Let's consider two tables, **CUSTOMERS** and **ORDERS**, to explain this:

- **CUSTOMERS Table (Parent Table)**: This table stores customer details, such as id, name, age, salary, and address. The **id** column is the **primary key,** which uniquely identifies each customer.

- **ORDERS Table (Child Table):** This table holds information about customer orders, with columns like ID, DATE, CUSTOMER_ID, and AMOUNT. The **CUSTOMER_ID** in the **ORDERS table** acts as a **Foreign Key**, linking it to the **id** in the **CUSTOMERS table**.

```
-- Creating the CUSTOMER table
CREATE TABLE CUSTOMERS (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    salary DECIMAL(10, 2),
    address VARCHAR(255)
);

-- Creating the ORDERS table with a foreign key reference to CUSTOMERS
CREATE TABLE ORDERS (
    ID INT NOT NULL,
    DATE DATETIME,
    CUSTOMER_ID INT,
    AMOUNT DECIMAL(10, 2),
    CONSTRAINT fk_customer
        FOREIGN KEY (CUSTOMER_ID)
        REFERENCES CUSTOMERS (id)
);
```

-- Inserting sample data into the CUSTOMERS table

```
INSERT INTO CUSTOMERS (id, name, age, salary, address)
VALUES
   (1, 'Alice Johnson', 30, 50000.00, '123 Main St'),
   (2, 'Bob Smith', 40, 60000.00, '456 Oak St');
```

-- Inserting sample data into the ORDERS table

```
INSERT INTO ORDERS (ID, DATE, CUSTOMER_ID, AMOUNT)
VALUES
   (101, '2024-09-01 10:00:00', 1, 150.00),
   (102, '2024-09-02 14:30:00', 2, 200.00);
```

How the Foreign Key Works:

The foreign key relationship ensures that every order is linked to a valid customer.

For example:

- Order ID 101 has CUSTOMER_ID = 1, which refers to Alice Johnson (in the CUSTOMERS table).
- Order ID 102 has CUSTOMER_ID = 2, which refers to Bob Smith.
- If you try to insert an order with a CUSTOMER_ID = 3 and there's no customer with id = 3 in the CUSTOMERS table, the database will reject the insertion , maintaining referential integrity.

2. Add a Foreign Key After Table Creation

Syntax-

ALTER TABLE child_table_name

ADD CONSTRAINT constraint_name

FOREIGN KEY (child_col1, child_col2)

REFERENCES parent_table_name (parent_col1, parent_col2);

Example- If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

ALTER TABLE ORDERS

ADD CONSTRAINT fk_customer

FOREIGN KEY (CUSTOMER_ID)

REFERENCES CUSTOMERS(id);

3. DROP a Constraint

Syntax-

ALTER TABLE table_name

DROP CONSTRAINT constraint_name;

Example- Suppose you have a foreign key constraint named fk_customer in the ORDERS table that references the CUSTOMERS table, and you want to drop this constraint.

ALTER TABLE ORDERS

DROP CONSTRAINT fk_customer;

LIKE Clause

- The LIKE clause is used in where clause to search the data using a specific pattern.

Syntax-

SELECT column_name(s)

FROM table_name

WHERE column_name LIKE pattern;

Wildcards used with LIKE:

1. **The percent sign (%):** Represents zero, one, or multiple characters.

Example: Find names starting with "K":

SELECT name

FROM CUSTOMERS

WHERE name LIKE 'K%';

name
Khilan
Kaushik

Example: Find names ending with "i":

SELECT name

FROM Employees

WHERE name LIKE '%i';

name
Chaitali
Bharti

Example: Find names that contain the letter "n" anywhere in the middle

```
SELECT name
FROM CUSTOMERS
WHERE name LIKE '%a%';
```

name
Ramesh
Khilan
Kaushik
Chaitali
Hardik
Bharti
Rahul

2. **The underscore (_):** Represents a single character.

Example: Find names with "a" as the second character:

```
SELECT name
FROM CUSTOMERS
WHERE name LIKE '_a%';
```

name
Ramesh
Kaushik
Hardik
Rahul

3. **The square bracket [ ]:** Specifies a range of characters.

Example: Find names starting with "C", "H":

```
SELECT name
FROM CUSTOMERS
WHERE name LIKE '[C-H]%';
```

name
Chaitali
Hardik

Set operators

```
create database set_operators
use set_operators

CREATE TABLE Students2
(Name VARCHAR(15),
TotalMark INT)

CREATE TABLE Students5
(St_Name VARCHAR(15),
TotalMark INT)

--lnserting data into student2

INSERT INTO Students2 VALUES ('Mahatma', 1070);
INSERT INTO Students2 VALUES ('Payal', 1032);
INSERT INTO Students2 VALUES ('Asish', 1002);
INSERT INTO Students2 VALUES('Debasish', 1063);
INSERT INTO Students2 VALUES ('Arpita', 1032);
INSERT INTO Students2 VALUES ('Chandan', 1034);


--Inserting data into student5
INSERT INTO Students5 VALUES('Rashmita', 1032);
INSERT INTO Students5 VALUES('Puja', 1086);
INSERT INTO Students5 VALUES ('Chandan', 1034);
INSERT INTO Students5 VALUES ('Priya', 1032);
INSERT INTO Students5 VALUES('saloni', 1032);
INSERT INTO Students5 VALUES('Arpita', 1032);

select * from students2
select * from students5
```

Name	TotalMark
Mahatma	1070
Payal	1032
Asish	1002
Debasish	1063
Arpita	1032
Chandan	1034

St_Name	TotalMark
Rashmita	1032
Puja	1086
Chandan	1034
Priya	1032
saloni	1032
Arpita	1032

1. **Union-** Union is used to combine the results of two queries into a single result set of all distinct rows. Both the queries must result in the same number of columns and compatible data types
in order to unite.

Syntax:-

```
SELECT * FROM TABLE1
UNION
SELECT * FROM TABLE2
```

Example-

```
select * from students2
union
select * from students5
```

Name	TotalMark
Arpita	1032
Asish	1002
Chandan	1034
Debasish	1063
Mahatma	1070
Payal	1032
Priya	1032
Puja	1086
Rashmita	1032
saloni	1032

2. **Union all**-UNION ALL is very similar to UNION, but with one exception: UNION ALL returns all the data from multiple select queries , no matter if it is a duplicate or not

Syntax:-

```
SELECT * FROM TABLE1
UNION ALL
SELECT * FROM TABLE2
```

Example-

```
select * from students2
union all
select * from students5
```

Name	TotalMark
Mahatma	1070
Payal	1032
Asish	1002
Debasish	1063
Arpita	1032
Chandan	1034
Rashmita	1032
Puja	1086
Chandan	1034
Priya	1032
saloni	1032
Arpita	1032

3. **Intersect-** Take the result of two queries and returns only those rows which are common in both result sets. It removes duplicate records from the final result set.

Syntax-

```
SELECT * FROM TABLE1
INTERSECT
SELECT * FROM TABLE2
```

Example-

```
select * from students2
intersect
select * from students5
```

Name	TotalMark
Arpita	1032
Chandan	1034

4. **EXCEPT:-** It is used to take the distinct records of two select queries and returns only those rows which does not appear in the second result set.

Syntax-

```
SELECT * FROM TABLE1
EXCEPT
SELECT * FROM TABLE2
```

Example-

```
select * from students2
except
select * from students5
```

Name	TotalMark
Asish	1002
Debasish	1063
Mahatma	1070
Payal	1032