4. Introduction to SQL

4.1 Background

- SQL is a standard database language used to access and manipulate data in databases.
- SQL stands for Structured Query Language. SQL was developed by IBM Computer Scientists in the 1970s.
- By executing queries SQL can create, update, delete, and retrieve data in databases like MySQL, Oracle, PostgreSQL, etc.
- Overall SQL is a query language that communicates with databases.

Disadvantages of SQL:

- ✓ SQL can be difficult to learn and master, especially for beginners.
- ✓ SQL is not well-suited for managing unstructured data or data that does not fit into a tabular format.
- ✓ SQL can be slow and resource-intensive for large data sets, especially when performing complex aggregation or join operations.

Advantages of SQL:

- SQL is a powerful and expressive language for managing relational databases.
- SQL supports data manipulation, data definition, and data control operations, making it a versatile tool for managing data.
- SQL is widely supported by most modern database management systems, making it a universal language for managing data.
- SQL is designed to work with large data sets, making it a powerful tool for data analysis and reporting.
- SQL supports a wide range of data types and data structures, making it a flexible tool for managing diverse data sets.
- SQL supports concurrent access and transactions, making it a reliable tool for managing data in multi-user environments.

4.2 Data Types in SQL

Basic data types in SQL with their features and examples:

- **Integer (INT):**
  - Integer data types are used to store whole numbers with no decimal points.

- They are typically stored as 4-byte integers and can store values between -2147483648 and 2147483647.
  Example: INT

- **Numeric (NUMERIC, DECIMAL):**
  - Numeric data types are used to store decimal numbers.
  - They can store up to 38 digits of precision and a scale of up to 38 digits.
  - Example: NUMERIC(5,2) (stores decimal numbers with 5 digits in total and 2 digits after the decimal point).
- **Float (FLOAT, REAL):**
  - Float data types are used to store decimal numbers with a floating point.
  - They can store up to 53 bits of precision and are typically stored as 8-byte floating point numbers.
  - Example: FLOAT (stores decimal numbers with a floating point).
- **Date and Time (DATE, TIME, TIMESTAMP):**
  - Date and time data types are used to store dates and times.
  - They can store values in various formats, such as 'YYYY-MM-DD' for date, 'HH:MM:SS' for time, and 'YYYY-MM-DD HH:MM:SS' for timestamp.
  - Example: DATE (stores date values in the format 'YYYY-MM-DD').
- **Character (CHAR, VARCHAR):**
  - Character data types are used to store strings of characters.
  - CHAR data types have a fixed length and can store up to 255 characters, while VARCHAR data types have a variable length and can store up to 65535 characters.
  - Example: CHAR(5) (stores strings of characters with a fixed length of 5 characters).
- **Text (TEXT, CLOB):**
  - Text data types are used to store large strings of characters.
  - They can store up to 2GB of text data and are typically used for storing large documents or text data.
  - Example: TEXT (stores large strings of characters).
- **Boolean (BOOLEAN, BIT):**
  - Boolean data types are used to store true or false values.
  - They can store values of true or false and are typically used for storing flags or switches.
  - Example: BOOLEAN (stores true or false values).

- **Binary (BINARY, VARBINARY, BLOB):**
  - Binary data types are used to store binary data such as images or audio files.
  - They can store up to 2GB of binary data and are typically used for storing multimedia files or other binary data.
  - Example: BINARY(10) (stores binary data with a fixed length of 10 bytes).

4.3 Types of SQL Commands:

1)Data Definition Language (DDL) Commands: DDL commands are used to create, modify, and delete database objects such as tables, schemas, and indexes.

- CREATE: Creates a new database object(table, database).
  Example: CREATE TABLE students (id INT, name VARCHAR(50), age INT);
- ALTER: Modifies an existing database object(table).
  Example: ALTER TABLE students ADD grade VARCHAR(50);
- DROP: Deletes an existing database object(table,database).
  Example: DROP TABLE students;

2)Data Manipulation Language (DML) Commands: DML commands are used to insert, update, and delete data in a database.

- INSERT: Inserts new data into a table.
  Example: INSERT INTO students (id, name, age) VALUES (1, 'John', 25);
- UPDATE: Updates existing data in a table.
  Example: UPDATE students SET age = 26 WHERE id = 1;
- DELETE: Deletes existing data from a table.
  Example: DELETE FROM students WHERE id = 1;

3)Data Query Language (DQL) Commands: DQL commands are used to retrieve data from a database.

- SELECT: Retrieves data from a table.
  Example: SELECT * FROM students;
- WHERE: Filters data based on a condition.
  Example: SELECT * FROM students WHERE age > 25;
- GROUP BY: Groups data based on a column.

Example: SELECT age, COUNT(*) FROM students GROUP BY age;
- ORDER BY: Orders data based on a column.
  Example: SELECT * FROM students ORDER BY age DESC;

4)Data Control Language (DCL) Commands: DCL commands are used to control access to a database and manage database privileges.

- GRANT: Grants access to a database object.
  Example: GRANT SELECT ON students TO user;
- REVOKE: Revokes access to a database object.
  Example: REVOKE SELECT ON students FROM user;

5)Transaction Control Language (TCL) Commands: TCL commands are used to manage database transactions and rollback or commit changes.

- BEGIN: Begins a new transaction.
  Example: BEGIN TRANSACTION;
- ROLLBACK: Rolls back a transaction and undoes changes.
  Example: ROLLBACK TRANSACTION;
- COMMIT: Commits a transaction and saves changes.
  Example: COMMIT TRANSACTION;

These are the basic types of SQL commands with their uses and examples. Different database management systems may support additional or modified commands, but these are the most commonly used and recognized SQL commands.

4.4 The basic structure of an SQL query:

SELECT select_list of columns

FROM schema_name.table_name. / TableName

WHERE conditions;

Here,

- **SELECT** is the SQL keyword that is used to **select data** from a database.
- The select_list can be a column name or a list of column names that you want to select, separated by commas.
- You can also use * to select all columns.
- **FROM** is the SQL keyword that indicates the table name from which data needs to be selected.

- **WHERE** is an optional keyword that is used to filter records . It is used to extract only those records that fulfill a specified condition.

1. SQL query that selects all columns from the employees table:

   SELECT * FROM employees;

2. You can also select specific columns like this:

   SELECT first_name, last_name, department

   FROM employees;

3. To filter records, you can use the WHERE clause like this:

   SELECT * FROM employees

   WHERE department = 'Sales';

   - This will select all columns from the employees table where the department column is equal to **'Sales'**.
   - You can use other comparison operators like **<, >, <=, >=,** and <> to filter records based on different conditions.

4. You can also use logical operators like AND, OR, and NOT to combine multiple conditions in the WHERE clause.
   For example:

   SELECT * FROM employees

   WHERE department = 'Sales' **AND** salary > 50000;

This will select all columns from the **employees** table where the **department** column is equal to **'Sales'** and the **salary** column is greater than **50000.**

4.5 Table Creation, Data Insertion, Data updating, Data Selection

Table Creation:

To create a table in SQL, you can use the CREATE TABLE statement. The basic **syntax** of the CREATE TABLE statement is as follows:

CREATE TABLE table_name

(

column1 data_type constraints,

column2 data_type constraints,

...);

Here,

- **CREATE TABLE** is the SQL keyword that is used to create a new table.
- **table_name** is the name of the table that you want to create.
- column1, column2, ... are the names of the columns that you want to create in the table.
- data_type is the data type of the column. Some common data types are INT, VARCHAR, DATE, etc.
- constraints are optional rules that you can apply to the columns to control the data that can be inserted into the table. Some common constraints are PRIMARY KEY, UNIQUE, NOT NULL, FOREIGN KEY, etc.

Here's an example of an SQL query that creates a table named employees with the following columns:

id: An integer that serves as the primary key for the table.

first_name: A variable-length character string that cannot be null.

last_name: A variable-length character string that cannot be null.

email: A variable-length character string that must be unique and cannot be null.

department: A variable-length character string that cannot be null.

salary: A decimal number that cannot be null.

hire_date: A date that cannot be null.

CREATE TABLE employees (

id INT PRIMARY KEY,

first_name VARCHAR(50) NOT NULL,

last_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

department VARCHAR(50) NOT NULL,

salary DECIMAL(10, 2) NOT NULL,

hire_date DATE NOT NULL);

Data Insertion:

In SQL, data insertion is the process of adding new rows to a table. This is typically done using the INSERT INTO statement.

The basic syntax for inserting data into a table in SQL is:

Syntax:

INSERT INTO table_name (column1, column2, ...)

VALUES

(value1, value2, ...);

In this syntax, table_name is the name of the table where you want to insert the new row. column1, column2, etc. are the names of the columns in the table where you want to insert values. value1, value2, etc. are the values that you want to insert into the corresponding columns.

For example, consider the following employees table:

```
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  1 | John Doe |  30 |
|  2 | Jane Doe |  25 |
+----+----------+-----+
```

To insert a new row into this table with the name "Bob Smith" and age 40, you would use the following INSERT INTO statement:

INSERT INTO employees (name, age) VALUES ('Bob Smith', 40);

```
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  1 | John Doe |  30 |
|  2 | Jane Doe |  25 |
|  3 | Bob Smith|  40 |
+----+----------+-----+
```

It's important to note that when inserting data into a table, you must ensure that the data types of the values match the data types of the corresponding columns. If the data types do not match, the INSERT statement will fail.

Data Updating:

In SQL, data updating is the process of modifying existing rows in a table. This is typically done using the UPDATE statement.

The basic syntax for updating data in a table in SQL is:

Syntax:

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

In this syntax, table_name is the name of the table where you want to update the existing rows. column1, column2, etc. are the names of the columns in the table where you want to update the values. value1, value2, etc. are the new values that you want to set for the corresponding columns. condition is a boolean expression that specifies which rows to update.

Example:

To update the **age of John Doe to 35**, you would use the following UPDATE statement:

```
UPDATE employees
```

SET age = 35

WHERE name = 'John Doe';

o/p:

```
+----+----------+-----+
| id | name     | age |
+----+----------+-----+
|  1 | John Doe |  35 |
|  2 | Jane Doe |  25 |
|  3 | Bob Smith|  40 |
+----+----------+-----+
```

It's important to note that when updating data in a table, you must ensure that the condition expression specifies a unique set of rows. If the condition expression is not specific enough, multiple rows may be updated unintentionally.

You can also update multiple columns at once by separating the column-value pairs with commas:

UPDATE employees

SET age = 36, name = 'John Doe Jr.'

WHERE id = 1;

This will update both the age and name columns for the row with id equal to 1.

Data Selection:

In SQL, data selection is the process of retrieving data from one or more tables in a database. This is typically done using the SELECT statement.

The basic syntax for selecting data from a table in SQL is:

SELECT column1, column2, ...

FROM table_name

WHERE condition;

In this syntax, column1, column2, etc. are the names of the columns in the table that you want to retrieve. table_name is the name of the table where you want to retrieve data from. condition is an optional boolean expression that specifies which rows to retrieve.

To retrieve all the rows from this table, you would use the following SELECT statement:

SELECT *

FROM employees;

This will return all columns and rows from the employees table.

To retrieve only the name and age columns from this table, you would use the following SELECT statement:

SELECT name, age

FROM employees;

You can also use WHERE clause to filter the rows based on a condition

SELECT *

FROM employees

WHERE age > 30;

4.6 Changing Table Structure:

In SQL, changing table structure refers to adding, modifying, or deleting columns in an existing table. This typically done using the **ALTER TABLE`**.

The basic syntax for adding a to a table in SQL is:

ALTER TABLE table_name

ADD column_name datatype;

For example, to add a new column salary of type INT to the employees table, you would use the following ALTER TABLE statement:

ALTER TABLE employees

ADD salary INT;

The basic syntax for modifying a column in a table in SQL is:

ALTER TABLE table_name

MODIFY column_name datatype;

For example, to change the data type of the salary column to DECIMAL(10,2) in the employees table, you would use the following ALTER TABLE statement:

ALTER TABLE employees

MODIFY salary DECIMAL(10,2);

The basic syntax for deleting a column in a table in SQL is:

ALTER TABLE table_name

DROP COLUMN column_name;

For example, to delete the salary column in the employees table, you would use the following ALTER TABLE statement:

ALTER TABLE employees

DROP COLUMN salary;

4.7 Where Clause, DISTINCT Clause, Using Column Aliases:

Where Clause:

- The WHERE clause in SQL is used to filter rows in a table based on a specified condition. It is used in the SELECT, UPDATE, DELETE, and INSERT statements.

The basic syntax for the WHERE clause is:

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

For example, to retrieve all the rows from the employees table where the age is greater than 30, you would use the following SELECT statement:

```
SELECT *
FROM employees
WHERE age > 30;
```

This will return all columns and rows from the employees table where the age is greater than 30.

- The WHERE clause can also be used with comparison operators such as =, <>, <, <=, >, >=, IN, NOT IN, BETWEEN, LIKE, IS NULL, IS NOT NULL, AND, OR, XOR, NOT and parentheses to make the condition more complex.

You can also use WHERE clause in UPDATE and DELETE statements to update or delete rows based on a specified condition.

For example, to update the age of all the employees in the employees table where the name **starts with 'J' to 35**, you would use the following UPDATE statement:

```
UPDATE employees
SET age = 35
WHERE name LIKE 'J%';
```

DISTINCT Clause

The DISTINCT clause in SQL is used to eliminate duplicate rows from a result set. It is used in the SELECT statement.

The basic syntax for the DISTINCT clause is:

SELECT DISTINCT column1, column2, ...

FROM table_name;

For example, to retrieve all the unique department_id values from the employees table, you would use the following SELECT statement:

SELECT DISTINCT department_id

FROM employees;

- It's important to note that the DISTINCT clause is applied to the entire result set, not just the columns listed after the DISTINCT keyword.
- You can also use DISTINCT clause with aggregate functions like COUNT(), SUM(), AVG(), MIN(), MAX() to perform calculations on the unique values.

For example, to retrieve the number of unique department_id values from the employees table, you would use the following SELECT statement:

SELECT COUNT(DISTINCT department_id)

FROM employees;

This will return the number of unique department_id values from the employees table.

Using Column Aliases

- In SQL, column aliases are used to give a column a temporary name, which can be different from its actual name, for the duration of a query.
- Column aliases are used to make the query results more readable and simplify the column names.

The basic syntax for using column aliases is:

SELECT column1 AS alias1, column2 AS alias2, ...

FROM table_name;

For example, to retrieve all the rows from the employees table and give the salary column the alias monthly_salary, you would use the following SELECT statement:

```
SELECT salary AS monthly_salary

FROM employees;
```

This will return all columns and rows from the employees table, but the salary column will be displayed as monthly_salary.

4.8 Working with Views

A view in SQL is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table.

The fields in a view are fields from one or more real tables in the database.

a) Creating View on table:

A view is created by using a SELECT statement, and it can be used to simplify complex queries, hide sensitive data, and provide a simpler interface for end-users.
The basic syntax for creating a view is:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

For example, to create a view that contains all the rows from the employees table where the salary is greater than 5000, you would use the following CREATE VIEW statement:

```
CREATE VIEW high_salary_employees AS
SELECT *
FROM employees
WHERE salary > 5000;
```

This will create a view called high_salary_employees that contains all the columns and rows from the employees table where the salary is greater than 5000.

b) Creating view on view:

In SQL, it is possible to create a view based on another view. This is known as a nested view or a view on a view.
The basic syntax for creating a view based on another view is:

CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM **view_name**;

For example, if you have a view called high_salary_employees that contains all the rows from the employees table where the salary is greater than 5000, you can create a new view
called **high_salary_employees_dept10** based on this view to filter the data based on a specified condition.

```
CREATE VIEW high_salary_employees_dept10 AS
SELECT *
FROM high_salary_employees
WHERE department_id = 10;
```

c) <u>Updating view:</u>

A view is a database object that can contain rows (all or selected) from an existing table. It can be created from one or many tables which depends on the provided SQL query to create a view.

Unlike **CREATE VIEW** and **DROP VIEW** there is no direct statement to update the records of an existing view. We can use the
SQL **UPDATE** Statement to modify the existing records in a table or a view.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows −

```
UPDATE view_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Following query creates a view based on the above created table −

CREATE VIEW CUSTOMERS_VIEW AS SELECT * FROM CUSTOMERS;

You can verify the contents of a view using the SELECT query as shown below −

SELECT * FROM CUSTOMERS_VIEW;

The view will be displayed 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

Following query updates the age of Ramesh to 35 in the above created CUSTOMERS_VIEW −

UPDATE CUSTOMERS_VIEW
SET AGE = 35 WHERE name = 'Ramesh';

You can verify the contents of the CUSTOMERS_VIEW using the SELECT statement as follows −

SELECT * FROM CUSTOMERS_VIEW WHERE NAME ='Ramesh';

The resultant view would have the following record(s) −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00

d) <u>Altering view:</u>

- In SQL, you can use the **ALTER VIEW** statement to modify the structure of an existing view.
- This statement allows you to change the definition of a view by adding, removing, or modifying columns, or changing the underlying query.

The basic syntax for altering a view is:

ALTER VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

For example, if you have a view called high_salary_employees that contains all the rows from the employees table where the salary is greater than 5000, and you want to add a new column hire_date to the view,

you would use the following ALTER VIEW statement:

ALTER VIEW high_salary_employees AS

SELECT *, hire_date

FROM employees

WHERE salary > 5000;

This will add a new column hire_date to the high_salary_employees view, which will contain the hire_date column from the employees table.

It's important to note that the ALTER VIEW statement can only be used to modify the structure of the view, it cannot be used to modify the data in the underlying tables.

It's also important to note that the ALTER VIEW statement is not supported by all SQL databases, some databases use different syntax to modify the structure of a view.

ALTER VIEW statement can also be used to rename a view by using the RENAME TO clause.

For example, if you have a view called **high_salary_employees** and you want to rename it to **high_paid_employees**, you would use the following ALTER VIEW statement:

```
ALTER VIEW high_salary_employees
RENAME TO high_paid_employees;
```

This will rename the high_salary_employees view to high_paid_employees.

It's important to note that the **ALTER VIEW** statement can also be used to drop a view by using the **DROP VIEW** clause.

For example, if you have a view called **high_paid_employees** and you want to drop it, you would use the following ALTER VIEW statement:

```
ALTER VIEW high_paid_employees
DROP VIEW;
```

This will drop the high_paid_employees view.

It's important to note that dropping a view will permanently delete it and all the data that it contains. It's important to make sure that you no longer need the view before dropping it.

4.9 SQL Functions:

- In SQL, a function is a set of statements that performs a specific task and returns a value.

- Functions are used to perform calculations, manipulate data, and return a single value.

- SQL provides a variety of built-in functions, and you can also create your own custom functions.

a) Single Row Functions:

In SQL, single-row functions are scalar functions that operate on a single row of data and return a single value.

These functions are used to perform calculations, manipulate data, and return a single value for a single row.

Examples of built-in single-row functions include:

DATE Functions:

In SQL, date functions are used to manipulate and format date and time data. SQL provides a variety of built-in date functions that can be used to extract, compare, and manipulate date and time data.

Here are some examples of built-in date functions:

NOW(), GETDATE(), CURRENT_DATE(): Returns the current date and time.

```
SELECT NOW();
```

CURDATE(), CURRENT_DATE(): Returns the current date.

```
SELECT CURDATE();
```

CURTIME(): Returns the current time.

```
SELECT CURTIME();
```

YEAR(date): Returns the year of a date.

```
SELECT YEAR('2022-03-01');
```

MONTH(date): Returns the month of a date.

```
SELECT MONTH('2022-03-01');
```

DAY(date): Returns the day of a date.

```
SELECT DAY('2022-03-01');
```

DATEDIFF(datepart, date1, date2): Returns the number of interval boundaries crossed between two dates.

```
SELECT DATEDIFF(DAY, '2022-03-01', '2022-03-08');
```

DATE_FORMAT(date, format): Returns the date in a specified format.

```
SELECT DATE_FORMAT('2022-03-01', '%Y-%m-%d %H:%i:%s');
```

STR_TO_DATE(string, format): Returns a date from a string.

```
SELECT STR_TO_DATE('01-MAR-2022', '%d-%b-%Y');
```

LAST_DAY(date): Returns the last day of the month for a given date.

```
SELECT LAST_DAY('2022-03-01');
```

EXTRACT(datepart FROM date): Returns a specific part of a date.

```
SELECT EXTRACT(YEAR FROM '2022-03-01');
```

It's important to note that the syntax for using date functions can vary between different SQL databases.

Additionally, it's a good practice to test the SQL statement on a development or test system before applying it to a production system.

You can also use date functions in
the SELECT clause, WHERE clause, ORDER BY clause, and HAVING clause.

For example, to retrieve all the rows from the employees table where
the hire_date is in March 2022, you can use
the MONTH() and YEAR() function in the WHERE clause:

```
SELECT *

FROM employees

WHERE MONTH(hire_date) = 3 AND YEAR(hire_date) = 2022;
```

This will return all the rows from the employees table where the hire_date is in March 2022.

Character Function:

- **LENGTH(string):** Returns the length of the string.

- **CONCAT(string1, string2, ...):** Returns the concatenation of two or more strings.
- **SUBSTRING(string, start, length):** Returns a substring of a string, starting at the specified position and having the specified length.
- **TRIM(string):** Returns the string with leading and trailing spaces removed.
- **LTRIM(string):** Returns the string with leading spaces removed.
- **RTRIM(string):** Returns the string with trailing spaces removed.

CASE Manipulation Functions:

- **UPPER(string):** Returns the string in uppercase.

```
SELECT UPPER(name)
FROM employees;
```

- **LOWER(string):** Returns the string in lowercase.

It's also important to note that some single-row functions can only be used in the SELECT clause, such as UPPER(), LOWER(),

LENGTH(), CONCAT(), SUBSTRING(), TRIM(), LTRIM(), RTRIM(), ABS( ), SQRT(), RAND(), IF(), CASE, etc.

Number Functions:

In SQL, number functions are used to perform mathematical operations on numerical data. SQL provides a variety of built-in number functions that can be used to perform mathematical operations, extract, compare, and manipulate numerical data.

Here are some examples of built-in number functions:

ABS(number): Returns the absolute value of a number.

```
SELECT ABS(-123);
```

CEIL(number): Returns the smallest integer greater than or equal to a number.

```
SELECT CEIL(123.45);
```

FLOOR(number): Returns the largest integer less than or equal to a number.

```
SELECT FLOOR(123.45);
```

ROUND(number, decimals): Rounds a number to a specified number of decimals.

```
SELECT ROUND(123.456, 2);
```

RAND(): Returns a random number between 0 and 1.

```
SELECT RAND();
```

POWER(number, power): Returns the result of a number raised to a power.

```
SELECT POWER(2, 3);
```

SQRT(number): Returns the square root of a number.

```
SELECT SQRT(16);
```

MOD(number1, number2): Returns the remainder of dividing number1 by number2.

```
SELECT MOD(10, 3);
```

Conversion Functions:

In SQL, conversion functions are used to convert data from one data type to another. SQL provides a variety of built-in conversion functions that can be used to convert data between different data types.

Here are some examples of built-in conversion functions:

CAST(expression AS data_type): Converts an expression to a specified data type.

```
SELECT CAST('123' AS UNSIGNED)
```

CONVERT(expression, data_type): Converts an expression to a specified data type.

```
SELECT CONVERT('123', UNSIGNED)
```

PARSE_DATE(format, string): Converts a string to a date using a specified format.

SELECT PARSE_DATE('%d-%b-%Y', '01-MAR-2022')

PARSE_TIME(format, string): Converts a string to a time using a specified format.

SELECT PARSE_TIME('%H:%i:%s', '13:30:00')

PARSE_DATETIME(format, string): Converts a string to a datetime using a specified format.

SELECT PARSE_DATETIME('%Y-%m-%d %H:%i:%s', '2022-03-01 13:30:00')

STR_TO_DATE(string, format): Converts a string to a date using a specified format.

SELECT STR_TO_DATE('01-MAR-2022', '%d-%b-%Y')

STR_TO_TIME(string, format): Converts a string to a time using a specified format.

SELECT STR_TO_TIME('13:30:00', '%H:%i:%s')

STR_TO_DATETIME(string, format): Converts a string to a datetime using a specified format.

SELECT STR_TO_DATETIME('2022-03-01 13:30:00', '%Y-%m-%d %H:%i:%s')

BIN(string): Converts a string to binary.

SELECT BIN('123')

UNHEX(string): Converts binary to a string.

SELECT UNHEX('7B')

CHAR(number): Converts a number to a character.

SELECT CHAR(50)

ASCII(string): Returns the ASCII code of the first character of a string.

SELECT ASCII('A')

Note: It's important to note that the syntax for using conversion functions can vary between different SQL databases.

b) Multiple Row Functions:

The multi-row function in SQL is used to retrieve data per set of rows at the time when we work on the group by clause we use the Multi-Row Function.

Types of Multi-Row Functions are

- Maximum(Max)
- Minimum(MIN)
- Average(Avg)
- Sum
- Count

MAX () Function in SQL:-

Returns the maximum value of a from the given data.

Syntax :

```
Max(expression)
```

For example:-

Select max (e_name)

from employees;
Output

5000

Display max salary of whose department_id=20

```
Select min(salary)

from employees where department_id=20;
```

Output → 3000

Multiple function using Max:-

```
Select max(salary +isnull(comm,0))

from emp
```

Output → 5000

Max _function on Character

```
Select max (e_name)

from employees
```

Output → word{based on Ascii}

Max _function on Date

```
Select max(hire_date)
from employees
```

Output → 1983

Min ( ) Function in SQL:-

Returns the minimum value of a given data.

Syntax:-

```
min(expression).
```

For Example:-

```
Select min(salary)
```

from employees

Output → 36025

Note:– We can not apply this function to varchar2 & data columns, Apply only on columns that have the numeric data.

display total salary paid to the manager

Select sum(sal) from emp Where job ‘manager’;

avg ( ) Function in SQL:-

It returns avg value of the given expression

Select avg (sal) from emp;

Output → 2250.725

Select ceiling (avg(sal)) from emp;

Output → 2251

Note:- This function also we can’t apply on varchar & date column**.**

Widget not in any sidebars

Count () Function in SQL:-

Count no of values present in a column

Select count(empno) from emp

Output → 16

count * () Function in SQ:-

Count () Function in SQ:-

Count no of values present in a column

Select count(empno) from emp

Output → 16

count * () Function in SQL:-

It returns no of records in the table

Select count (*) from emp

Output → 16