1.What is database?

Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways.

Example: School Management Database, Bank Management Database.

2.What is table?

A relational database system contains one or more objects called tables. The data or information for the database are stored in these tables. Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. Rows contain the records or data for the columns. Here is a sample table called "weather",city, state, high, and low are the columns. The rows contain the data for this table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Weather** | | | |
| **city** | **state** | **high** | **low** |
| Phoenix | Arizona | 105 | 90 |
| Tucson | Arizona | 101 | 92 |
| Flagstaff | Arizona | 88 | 69 |
| San Diego | California | 77 | 60 |
| Albuquerque | New Mexico | 80 | 72 |

3.What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

+-----------+

| ADDRESS |

+-----------+

| Ahmedabad |

| Delhi |

| Kota |

| Mumbai |

| Bhopal |

| MP |

| Indore |

+----+------+

4. What is row?

A record, also called a row of data, is each individual entry that exists in a table. For example there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

A record is a horizontal entity in a table.

5. What is inner join?

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

The basic syntax of **INNER JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

ON table1.common\_field = table2.common\_field;

Example:

Consider the following two tables, (a) 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 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

| OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using INNER JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

INNER JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

6.What is Left outer join with example?

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

## Syntax:

The basic syntax of **LEFT JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

LEFT JOIN table2

ON table1.common\_field = table2.common\_field;

Here given condition could be any given expression based on your requirement

Now, let us join these two tables using LEFT JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+----+----------+--------+---------------------+

7.What is right outer join?

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

## Syntax:

The basic syntax of **RIGHT JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

RIGHT JOIN table2

ON table1.common\_field = table2.common\_field;

Now, let us join these two tables using RIGHT JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

8.example of max.sum,avg?

## The AVG() Function

SQL **AVG** function is used to find out the average of a field in various records.

To understand **AVG** function, consider an **employee\_tbl** table, which is having the following records:

SQL> SELECT \* FROM employee\_tbl;

+------+------+------------+--------------------+

| id | name | work\_date | daily\_typing\_pages |

+------+------+------------+--------------------+

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+------+------+------------+--------------------+

Now suppose based on the above table you want to calculate average of all the dialy\_typing\_pages, then you can do so by using the following command:

SQL> SELECT AVG(daily\_typing\_pages)

-> FROM employee\_tbl;

+-------------------------+

| AVG(daily\_typing\_pages) |

+-------------------------+

| 230.0000 |

+-------------------------+

You can take average of various records set using **GROUP BY** clause. Following example will take average all the records related to a single person and you will have average typed pages by every person.

SQL> SELECT name, AVG(daily\_typing\_pages)

-> FROM employee\_tbl GROUP BY name;

+------+-------------------------+

| name | AVG(daily\_typing\_pages) |

+------+-------------------------+

| Jack | 135.0000 |

| Jill | 220.0000 |

| John | 250.0000 |

| Ram | 220.0000 |

| Zara | 325.0000 |

+------+-------------------------+

## The MAX() Function

SQL **MAX** function is used to find out the record with maximum value among a record set.

To understand **MAX**function, consider an **employee\_tbl** table, which is having the following records:

SQL> SELECT \* FROM employee\_tbl;

+------+------+------------+--------------------+

| id | name | work\_date | daily\_typing\_pages |

+------+------+------------+--------------------+

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+------+------+------------+--------------------+

7 rows in set (0.00 sec)

Now suppose based on the above table you want to fetch maximum value of daily\_typing\_pages, then you can do so simply using the following command:

SQL> SELECT MAX(daily\_typing\_pages)

-> FROM employee\_tbl;

+-------------------------+

| MAX(daily\_typing\_pages) |

+-------------------------+

| 350 |

+-------------------------+

You can find all the records with maxmimum value for each name using **GROUP BY** clause as follows:

SQL> SELECT id, name, MAX(daily\_typing\_pages)

-> FROM employee\_tbl GROUP BY name;

+------+------+-------------------------+

| id | name | MAX(daily\_typing\_pages) |

+------+------+-------------------------+

| 3 | Jack | 170 |

| 4 | Jill | 220 |

| 1 | John | 250 |

| 2 | Ram | 220 |

| 5 | Zara | 350 |

+------+------+-------------------------+

You can use **MIN** Function along with **MAX** function to find out minimum value as well. Try out the following example:

SQL> SELECT MIN(daily\_typing\_pages) least, MAX(daily\_typing\_pages) max

-> FROM employee\_tbl;

+-------+------+

| least | max |

+-------+------+

| 100 | 350 |

+-------+------+

1 row in set (0.01 sec)

## The SUM() Function

SQL **SUM** function is used to find out the sum of a field in various records.

To understand **SUM**function, consider an **employee\_tbl** table, which is having the following records:

SQL> SELECT \* FROM employee\_tbl;

+------+------+------------+--------------------+

| id | name | work\_date | daily\_typing\_pages |

+------+------+------------+--------------------+

| 1 | John | 2007-01-24 | 250 |

| 2 | Ram | 2007-05-27 | 220 |

| 3 | Jack | 2007-05-06 | 170 |

| 3 | Jack | 2007-04-06 | 100 |

| 4 | Jill | 2007-04-06 | 220 |

| 5 | Zara | 2007-06-06 | 300 |

| 5 | Zara | 2007-02-06 | 350 |

+------+------+------------+--------------------+

Now suppose based on the above table you want to calculate total of all the dialy\_typing\_pages, then you can do so by using the following command:

SQL> SELECT SUM(daily\_typing\_pages)

-> FROM employee\_tbl;

+-------------------------+

| SUM(daily\_typing\_pages) |

+-------------------------+

| 1610 |

+-------------------------+

You can take sum of various records set using **GROUP BY** clause. Following example will sum up all the records related to a single person and you will have total typed pages by every person.

SQL> SELECT name, SUM(daily\_typing\_pages)

-> FROM employee\_tbl GROUP BY name;

+------+-------------------------+

| name | SUM(daily\_typing\_pages) |

+------+-------------------------+

| Jack | 270 |

| Jill | 220 |

| John | 250 |

| Ram | 220 |

| Zara | 650 |

+------+-------------------------+

9.Example of group by?The SQL **GROUP BY**clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2

FROM table\_name

WHERE [ conditions ]

GROUP BY column1, column2

ORDER BY column1, column2

Example:

Consider the CUSTOMERS table is having the following records:

+----+----------+-----+-----------+----------+

| 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 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result:

+----------+-------------+

| NAME | SUM(SALARY) |

+----------+-------------+

| Chaitali | 6500.00 |

| Hardik | 8500.00 |

| kaushik | 2000.00 |

| Khilan | 1500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 2000.00 |

+----------+-------------+

Now, let us have following table where CUSTOMERS table has the following records with duplicate names:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Ramesh | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | kaushik | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result:

+---------+-------------+

| NAME | SUM(SALARY) |

+---------+-------------+

| Hardik | 8500.00 |

| kaushik | 8500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 3500.00 |

10.Example of having?

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

## Syntax:

The following is the position of the HAVING clause in a query:

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| 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 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is the example, which would display record for which similar age count would be more than or equal to 2:

SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM CUSTOMERS

GROUP BY age

HAVING COUNT(age) >= 2;

This would produce the following result:

+----+--------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+--------+-----+---------+---------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

+----+--------+-----+---------+---------+

11.Example of where?

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

You can specify a condition using [comparison or logical operators](https://www.tutorialspoint.com/sql/sql-operators.htm) like >, <, =, LIKE, NOT, etc. Below examples would make this concept clear.

Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| 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 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with name **Hardik**. Here, it is important to note that all the strings should be given inside single quotes ('') where as numeric values should be given without any quote as in above example:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME = 'Hardik';

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 5 | Hardik | 8500.00 |

+----+----------+----------+

12) Primary Key.

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**.

If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s).

**Note:** You would use these concepts while creating database tables.

Create Primary Key:

Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax:

ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);

**NOTE:** If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID, NAME)

);

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax:

ALTER TABLE CUSTOMERS

ADD CONSTRAINT PK\_CUSTID PRIMARY KEY (ID, NAME);

Delete Primary Key:

You can clear the primary key constraints from the table, Use Syntax:

ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;

13) A foreign key is a key used to link two tables together. This is sometimes called a referencing key.

Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

**The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.**

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

## Example:

Consider the structure of the two tables as follows:

CUSTOMERS table:

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

ORDERS table:

CREATE TABLE ORDERS (

ID INT NOT NULL,

DATE DATETIME,

CUSTOMER\_ID INT references CUSTOMERS(ID),

AMOUNT double,

PRIMARY KEY (ID)

);

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

ALTER TABLE ORDERS

ADD FOREIGN KEY (Customer\_ID) REFERENCES CUSTOMERS (ID);

## DROP a FOREIGN KEY Constraint:

To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE ORDERS