**What is Database?**

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content.

**What is Table?**

In relational database terms, a table is responsible for storing data in the database. Database tables consist of rows and columns.

**What is Column?**

Columns run vertically. They contain the definition of each field.

You give each column a name, so that it is describes the data that is stored. Examples of column names could include FirstName, LastName, ProductId, Price, etc

**What is row?**

Rows run horizontally. They represent each record. A row is the smallest unit of data that can be inserted into a database.

Rows span multiple columns, and therefore, the definition of a column applies to the cell where the row intersects with that column.

**Example for Inner join:**

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

Left table:

And a selection from the "Individual" table:

|  |  |  |  |
| --- | --- | --- | --- |
| IndividualID | FirstName | LastName | Username |
| 1 | Fred | Flinstone | Freddo |
| 2 | Homer | Simpson | Homey |
| 3 | Homer | Brown | Notsofamus |
| 4 | Ozzy | Ozzbourne | sabbath |
| 5 | Homer | Gain | noplacelike |

Right Table:

And a selection from the "Publisher" table:

|  |  |
| --- | --- |
| IndividualID | AccessLevel |
| 1 | Administrator |
| 2 | Contributor |
| 3 | Contributor |
| 4 | Contributor |
| 10 | Administrator |

Statement:

SELECT \* FROM Individual

INNER JOIN Publisher

ON Individual.IndividualID = Publisher.IndividualID

WHERE Individual.IndividualId = '2';

Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IndividualID | FirstName | LastName | UserName | IndividualID | AccessLevel |
| 2 | Homer | Simpson | homey | 2 | Contributor |

**Example for Left outer join:**

LEFT JOIN performs a join starting with the first (left-most) table and then any matching second (right-most) table records.

LEFT JOIN and LEFT OUTER JOIN are the same.

General Syntax:

SELECT column-names

FROM table-name1 LEFT JOIN table-name2

ON column-name1 = column-name2

WHERE condition

Example:

SELECT \* FROM Individual AS Ind

LEFT JOIN Publisher AS Pub

ON Ind.IndividualId = Pub.IndividualId;

Left table:

And a selection from the "Individual" table:

|  |  |  |  |
| --- | --- | --- | --- |
| IndividualID | FirstName | LastName | Username |
| 1 | Fred | Flinstone | Freddo |
| 2 | Homer | Simpson | Homey |
| 3 | Homer | Brown | Notsofamus |
| 4 | Ozzy | Ozzbourne | sabbath |
| 5 | Homer | Gain | noplacelike |

Right Table:

And a selection from the "Publisher" table:

|  |  |
| --- | --- |
| IndividualID | AccessLevel |
| 1 | Administrator |
| 2 | Contributor |
| 3 | Contributor |
| 4 | Contributor |
| 10 | Administrator |

Statement:

SELECT \* FROM Individual

INNER JOIN Publisher

ON Individual.IndividualID = Publisher.IndividualID

WHERE Individual.IndividualId = '2';

**Result:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IndividualID | FirstName | LastName | UserName | IndividualID | AccessLevel |
| 1 | Fred | Flinstone | Freddo | 1 | Administrator |
| 2 | Homer | Simpson | Homey | 2 | Contributor |
| 3 | Homer | Brown | Notsofamus | 3 | Contributor |
| 4 | Ozzy | Osbourne | Noplacelike | Null | Null |

**Example for Right outer join:**

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL Statement:

SELECT \* FROM Individual AS Ind

RIGHT JOIN Publisher AS Pub

ON Ind.IndividualId = Pub.IndividualId;

Left table:

And a selection from the "Individual" table:

|  |  |  |  |
| --- | --- | --- | --- |
| IndividualID | FirstName | LastName | Username |
| 1 | Fred | Flinstone | Freddo |
| 2 | Homer | Simpson | Homey |
| 3 | Homer | Brown | Notsofamus |
| 4 | Ozzy | Ozzbourne | sabbath |
| 5 | Homer | Gain | noplacelike |

Right Table:

And a selection from the "Publisher" table:

|  |  |
| --- | --- |
| IndividualID | AccessLevel |
| 1 | Administrator |
| 2 | Contributor |
| 3 | Contributor |
| 4 | Contributor |
| 10 | Administrator |

**Result:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IndividualID | FirstName | LastName | UserName | IndividualID | AccessLevel |
| 1 | Fred | Flinstone | Freddo | 1 | Administrator |
| 2 | Homer | Simpson | Homey | 2 | Contributor |
| 3 | Homer | Brown | Notsofamus | 3 | Contributor |
| 4 | Ozzy | Osbourne | Noplacelike | 4 | Contributor |
| Null | Null | Null | Null | 10 | Administrator |

**Example for Max, sun, Avg :**

**The SQL MAX syntax**

SELECT MAX(column-name)

FROM table-name

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

Select MAX(SALARY)

From CUSTOMERS;

Result:

|  |
| --- |
| MAX(SALARY) |
| 10000 |

**The SQL SUM syntax:**

SELECT SUM returns the sum of the data values.

**General syntax for SUM:**

SELECT SUM(column-name)

FROM table-name

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

Select SUM(SALARY)

From CUSTOMERS;

Result:

|  |
| --- |
| SUM(SALARY) |
| 35000 |

**The SQL SUM syntax:**

SELECT AVG returns the average of the data values

The General Syntax:

SELECT AVG (Column Name)

FROM table name

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

Select SUM(SALARY)

From CUSTOMERS;

Result:

|  |
| --- |
| AVG(SALARY) |
| 500 |

**Example of where Clause:**

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

**General syntax for where clause:**

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

Select \*

From CUSTOMERS

Where age >= 27;

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 5 | Hardik | 27 | Bhopal | 8500 |

**Example of Having:**

HAVING filters records that work on summarized GROUP BY results.

HAVING applies to summarized group records, whereas WHERE applies to individual records.

Only the groups that meet the HAVING criteria will be returned.

HAVING requires that a GROUP BY clause is present.

WHERE and HAVING can be in the same query.

The SQL HAVING syntax:

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

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 |

**Example for Primary key:**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values.

A primary key column cannot contain NULL values.

Most tables should have a primary key, and each table can have only ONE primary key

SQL Primary Key Table Creation:

CREATE TABLE Persons

(

P\_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (P\_Id)

)

**Foreign Key:**

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

And a selection from the "Individual" table:

Customers Table:

|  |  |  |  |
| --- | --- | --- | --- |
| CustomerID | FirstName | LastName | Username |
| 1 | Fred | Flinstone | Freddo |
| 2 | Homer | Simpson | Homey |
| 3 | Homer | Brown | Notsofamus |
| 4 | Ozzy | Ozzbourne | sabbath |
| 5 | Homer | Gain | noplacelike |

Orders Table:

And a selection from the "Individual" table:

|  |  |  |
| --- | --- | --- |
| OrderID | OrderNo | CustomerID |
| 1 | 1234 | 2 |
| 2 | 4765 | 1 |
| 3 | Homer | 5 |
| 4 | Ozzy | 1 |
| 5 | Homer | 3 |

Note that the "CustomerID" column in the "Orders" table points to the "CustomersID" column in the "Customers" table.

The "CustomerID" column in the "Persons" table is the PRIMARY KEY in the "Customers" table.

The "CustomerID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it must be one of the values contained in the table it points to.

**Finding second highest salary from row table?**

Consider the CUSTOMERS table having the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | AGE | ADDRESS | SALARY |
| 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | Khilan | 25 | Delhi | 1500 |
| 3 | Kaushik | 23 | Kota | 2000 |
| 4 | Chaitali | 25 | Mumbai | 6500 |
| 5 | Hardik | 27 | Bhopal | 8500 |
| 6 | Komal | 22 | Madya Pradesh | 4500 |
| 7 | Muffy | 25 | Indore | 10000 |

SELECT MAX(Salary) FROM CUSTOMERS

WHERE Salary NOT IN (SELECT MAX(Salary) FROM CUSTOMERS)