**DATABASE**

**1.** **What is a database?**

* It is defined as collection of information/data which is organized in such a way that it can be accessed, managed and updated.
* Information can be in different formats like text, graphic, audio and so on.

Types of databases:

1. Relational: most common type of databases where all the information is stored in the form of tables and they are related to each other with key fields. For instance – Oracle, Sybase and Microsoft SQL Server.

2. Operational: commonly used by organizations where huge amount of information such as inventory management, purchases and transactions is updated and modified depending on the company.

3. Database warehouse: this is used to store past data, which doesn’t require any modifications or alterations and then comparing that with present data to determine the key trends.

4. Distributed: This type can be helpful for people who have their companies or business in different locations. So, as to share the common data by all the groups.

5. End-User: It can be used when the data which is in the form of pdf files, spreadsheets, presentations, word files, note pad, download files. All these small database form a different type of database called End-User.

**2.** **What is a table?**

* It is a data structure used to organize the information in the form of rows and columns.
* The data can be used to both store and display in specified format.

**3.** **What is a row?**

* It is a series of data which is laid in a horizontal fashion in the table. It is also called as a record or tuple.

**4.** **What is a Column?**

* It is a series of data which is laid in a vertical fashion in the table. One column might require a unique identifier.

**5.** **Example of inner join?**

Join: It’s an SQL instruction which combines data of 2 different tables based on the common field between them.

Inner join: It returns all the rows from multiple tables where the join condition is met.

**Example:**

1. Students

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Age** | **State** |
| 1 | Sowmya | 25 | Texas |
| 2 | Roja | 26 | California |
| 3 | Pallavi | 23 | Missouri |

2. Jobs

|  |  |  |  |
| --- | --- | --- | --- |
| **JobID** | **Date** | **S\_Id** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$/hr |
| 13 | 02-15-2017 | 2 | 52S/hr |
| 14 | 03-01-2017 | 3 | 55$/hr |

SQL> SELECT Id, Name, Amount, Date  
 From Students  
 Inner join Jobs  
 on Students.Id = Jobs.S\_Id;

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Amount** | **Date** |
| 3 | Pallavi | 45$/hr | 01-10-2016 |
| 2 | Roja | 52S/hr | 02-15-2017 |
| 3 | Pallavi | 55$/hr | 03-01-2017 |

**6.** **Example of left outer join?**

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

**Example:**

1. Students

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Age** | **State** |
| 1 | Sowmya | 25 | Texas |
| 2 | Roja | 26 | California |
| 3 | Pallavi | 23 | Missouri |

2. Jobs

|  |  |  |  |
| --- | --- | --- | --- |
| **JobID** | **Date** | **S\_Id** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$/hr |
| 13 | 02-15-2017 | 2 | 52S/hr |
| 14 | 03-01-2017 | 3 | 55$/hr |

SQL> SELECT Id, Name, Amount, Date  
 From Students  
 left join Jobs  
 on Students.Id = Jobs.S\_Id;

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Amount** | **Date** |
| 1 | Sowmya | null | null |
| 2 | Roja | 52S/hr | 02-15-2017 |
| 3 | Pallavi | 45$/hr | 01-10-2016 |
| 3 | Pallavi | 55$/hr | 03-01-2017 |

**7.** **Example of 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.

**Example:**

1.Students

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Age** | **State** |
| 1 | Sowmya | 25 | Texas |
| 2 | Roja | 26 | California |
| 3 | Pallavi | 23 | Missouri |

2. Jobs

|  |  |  |  |
| --- | --- | --- | --- |
| **JobID** | **Date** | **S\_Id** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$/hr |
| 13 | 02-15-2017 | 2 | 52S/hr |
| 14 | 03-01-2017 | 3 | 55$/hr |

SQL> SELECT Id, Name, Amount, Date  
 From Students  
 right join Jobs  
 on Students.Id = Jobs.S\_Id;

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Amount** | **Date** |
| 3 | Pallavi | 45$/hr | 01-10-2016 |
| 2 | Roja | 52S/hr | 02-15-2017 |
| 3 | Pallavi | 55$/hr | 03-01-2017 |

**8.** **Example of max,sum and avg?**

Jobs

|  |  |  |
| --- | --- | --- |
| **JobID** | **Date** | **Amount** |
| 12 | 01-10-2016 | 45$/hr |
| 13 | 02-15-2017 | 52S/hr |
| 14 | 03-01-2017 | 55$/hr |

**Max**

SELECT MAX(Amount) FROM Jobs  
 WHERE Date BETWEEN ‘01-05-2016’ AND ‘03-05-2017’

Output :

**Amount**

55$/hr

**Sum**

SELECT SUM(Amount) FROM Jobs  
WHERE Date BETWEEN ‘01-05-2016’ AND ‘03-05-2017’

Output**:**

**Amount**

152$/hr

**Avg**

SELECT AVG(Amount) FROM Jobs  
WHERE Date BETWEEN ‘01-05-2016’ AND ‘03-05-2017’

Output:

**Amount**

51$/hr

**9. Example for Group by**

The GROUP BY statement is used with an aggregate function (like: count, max, min, sum, avg) to group the result-set by one or more columns.

**Example:**

1.Customers

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **City** | **State** |
| 1 | Sowmya | Lewisville | Texas |
| 2 | Roja | Redmond | California |
| 3 | Pallavi | Maryville | Missouri |
| 4 | Rajesh | Houston | Texas |
| 5 | Mahesh | New Orleans | LA |
| 6 | Vrundha | Fremont | California |
| 7 | Leela | Cleveland | Ohio |

2. Orders

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Date** | **CustomerId** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$ |
| 13 | 02-15-2017 | 2 | 52$ |
| 12 | 03-01-2017 | 3 | 55$ |
| 14 | 03-23-2017 | 5 | 60$ |

1. **Group by with count:**

SELECT COUNT(Id), State

FROM Customers

GROUP BY State

HAVING COUNT(Id) > 1

**Output:**

|  |  |
| --- | --- |
| **COUNT(Id)** | **State** |
| 2 | Texas |
| 2 | California |

1. **Group by with sum:**

SELECT SUM(Amount), OrderId

FROM Orders

GROUP BY OrderId

|  |  |
| --- | --- |
| **SUM(Amount)** | **OrderId** |
| 12 | 100$ |
| 13 | 52$ |
| 14 | 60$ |

1. **Group by with max:**

SELECT Max(Amount), OrderId

FROM Orders

GROUP BY OrderId

|  |  |
| --- | --- |
| **Max(Amount)** | **OrderId** |
| 12 | 100$ |

1. **Group by with min:**

SELECT Min(Amount), OrderId

FROM Orders

GROUP BY OrderId

|  |  |
| --- | --- |
| **Min(Amount)** | **OrderId** |
| 13 | 52$ |

1. **Group by with avg:**

SELECT Avg(Amount), OrderId

FROM Orders

GROUP BY OrderId

|  |  |
| --- | --- |
| **Avg(Amount)** | **OrderId** |
| 12 | 50$ |
| 13 | 52$ |
| 14 | 60$ |

**10. Example for Having**

A HAVING clause in SQL specifies that an SQL SELECT statement should only return rows where aggregate values meet the specified conditions. It was added to the SQL language because the WHERE keyword could not be used with aggregate functions.

**Note:** It is only used with Select

**Example:**

1.Customers

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **City** | **State** |
| 1 | Sowmya | Lewisville | Texas |
| 2 | Roja | Redmond | California |
| 3 | Pallavi | Dallas | Texas |
| 4 | Rajesh | Houston | Texas |
| 5 | Mahesh | New Orleans | LA |
| 6 | Vrundha | Fremont | California |
| 7 | Leela | Cleveland | Ohio |

**Syntax:**

SELECT COUNT(Id), State

FROM Customers

GROUP BY State

HAVING COUNT(Id) > 1

**Output:**

|  |  |
| --- | --- |
| **COUNT(Id)** | **State** |
| 3 | Texas |
| 2 | California |

**11.Example for Where condition**

The WHERE clause is used to extract only those records that fulfill a specified condition.

**Note:** It is only used with Select, Insert, Update, Delete

**Example:**

Orders

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Date** | **CustomerId** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$ |
| 13 | 02-15-2017 | 2 | 52$ |
| 12 | 03-01-2017 | 3 | 55$ |
| 14 | 03-23-2017 | 5 | 60$ |

**Syntax:**

SELECT \* FROM Orders

WHERE OrderID=12;

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Date** | **CustomerId** | **Amount** |
| 12 | 01-10-2016 | 3 | 45$ |
| 12 | 03-01-2017 | 3 | 55$ |

**12.Example for Primary key**

A primary key is a special relational database table column (or combination of columns) designated to uniquely identify all table records. A primary key's main features are: It must contain a unique value for each row of data. It cannot contain null values.

**MySQL:**

CREATE TABLE SaibersysTeam (

SaibersysID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

PRIMARY KEY (SaibersysID) );

**Oracle/SQL Server/Ms Access:**

CREATE TABLE SaibersysTeam (

SaibersysID int NOT NULL PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

);

**13.Example for Foreign key**

A FOREIGN KEY is a key used to link two tables together. Foreign Key in a table points to a PRIMARY KEY in another table.

**MySQL:**

CREATE TABLE AmensysTeam (

AmensysID int NOT NULL,

AmensysNumber int NOT NULL,

SaibersysID int,

PRIMARY KEY (AmensysID),

FOREIGN KEY (SaibersysID) REFERENCES SaibersysTeam(SaibersysID)

);

**Oracle/SQL Server/Ms Access:**

CREATE TABLE AmensysTeam (

AmensysID int NOT NULL PRIMARY KEY,

AmensysNumber int NOT NULL,

SaibersysID int FOREIGN KEY REFERENCES SaibersysTeam (SaibersysID)

);

**14.Finding second highest salary from row table**

**Students**

|  |  |  |  |
| --- | --- | --- | --- |
| **StudentID** | **Name** | **Age** | **Salary** |
| 501 | Sowmya | 25 | 50$/hr |
| 502 | Pallavi | 23 | 52$/hr |
| 503 | Roja | 26 | 55$/hr |
| 504 | Vrundha | 25 | 60$/hr |

SELECT MAX(salary) FROM Students WHERE Salary NOT IN ( SELECT Max(Salary) FROM Students) as HighestSalary

**Output:**

**HighestSalary**

55$/hr

**15. What is stored procedure, user defined functions**

**16. What is index and how it will improve performance**

**17. What is ETL testing**

**18. Different ETL jobs (SSIS jobs, informatcia jobs)**

**19.Tables for employee management system  
  
20.Tables for e-commerce management system  
  
21.Tables for Library management system**