**What is Database?**

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it. Computer databases typically contain aggregations of data records or files, such as sales transactions, product catalogs and inventories, and customer profiles.

**What is Table?**

A relational database is made up of several components, of which the table is most significant. The database table is where all the data in a database is stored, and without tables, there would not be much use for relational databases.

A database consists of one or more tables. Each table is made up of rows and columns. If you think of a table as a grid, the column go from left to right across the grid and each entry of data is listed down as a row. Each row in a relational is uniquely identified by a primary key. This can be by one or more sets of column values. In most scenarios it is a single column, such as employeeID. Every relational table has one primary key. Its purpose is to uniquely identify each row in the database. No two rows can have the same primary key value. The practical result of this is that you can select every single row by just knowing its primary key.

**What is Column?**

Columns are defined to hold a specific type of data, such as dates, numeric, or textual data. In the simplest of definitions a column is defined by its name and data type. The name is used in SQL statements when selecting and ordering data, and the data type is used to validate information stored. So, a DateOfBirth column defined as a date, can be referred to in an order by clause as ORDER BY DateOfBirth

And, if you try to add a value of “Hello Kitty” to the column, as part of its validation, it will recognize it isn’t a date, and reject it. Columns names can’t be duplicated in a table. So, having two “name” columns is a no no. Though you could have two “name” columns, such as name1, and name2, you’ll learn later on, that this is frowned up, as it breaks normal form (I explain this in another post).

**What is Row?**

A table can contain zero or more rows. When there are zero, it said to be empty. There is not practical limit on the number of rows a table can hold; however, remember the table’s primary key may have some influence on this. What I mean, is that if your table holds states, and the primary key is the state’s abbreviation, then by definition, since there are only fifty states in the union, and you can not have duplicates in a primary key, your table is limited to fifty rows.There is no guarantee that the rows in a table are stored in a particular order. Use the ORDER BY clause to do so.

Also, strictly speaking, in a relational database there is no first or last row. Yes, you can tease out a first row of a result using a keyword such as LIMIT or TOP, but those are used once the data is retrieved and sorted. The difference here is that you’re seeing the first row of the result, not what is physically stored in the table.

**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.

**Example:**

SELECT Employees.FirstName, Orders.OrderID

FROM Employees

INNER JOIN Orders

ON Employees.EmployeeID=Orders.EmployeeID

ORDER BY Orders.OrderID;

**Example for 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:**

SELECT Employees.FirstName, Orders.OrderID

FROM Employees

LEFT JOIN Orders

ON Employees.EmployeeID=Orders.EmployeeID

ORDER BY Employees.FirstName;

**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.

**Example:**

SELECT Employees.FirstName, Orders.OrderID

FROM Employees

RIGHT JOIN Orders

ON Employees.EmployeeID=Orders.EmployeeID

ORDER BY Employees.FirstName;

**Example for MAX, SUM, AVG?**

**MAX:** The MAX() function returns the largest value of the selected column.

**Syntax:** SELECT MAX(column\_name) FROM table\_name;

**Example:**

SELECT MAX(Price) AS HighestPrice FROM Products;

**SUM:** The SUM() function returns the total sum of a numeric column.

**Syntax:** SELECT SUM(column\_name) FROM table\_name;

**Example:**

SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;

**AVG:** The AVG() function returns the average value of a numeric column.

**Syntax:** SELECT AVG(column\_name) FROM table\_name

**Example:**

SELECT AVG(Price) AS PriceAverage FROM Products;

**Example for Group By?**

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

**Syntax:**

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name;

**Example:**

SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders

LEFT JOIN Shippers

ON Orders.ShipperID=Shippers.ShipperID

GROUP BY ShipperName;

**Example for 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:**

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

FROM CUSTOMERS

GROUP BY age

HAVING COUNT(age) >= 2;

**Example for Where condition?**

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

**SYNTAX:**

SELECT column\_name,column\_name

FROM table\_name

WHERE column\_name operator value;

**Example:**

SELECT \* FROM Customers

WHERE Country='Mexico';

**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.

**Example:**

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))

**Example for Foreign key?**

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

Let's illustrate the foreign key with an example. Look at the following two tables:

**The "Persons" table:**

P\_Id LastName FirstName Address City

1 Hansen Ola Timoteivn 10 Sandnes

2 Svendson Tove Borgvn 23 Sandnes

3 Pettersen Kari Storgt 20 Stavanger

**The "Orders" table:**

O\_Id OrderNo P\_Id

1 77895 3

2 44678 3

3 22456 2

4 24562 1

Note that the "P\_Id" column in the "Orders" table points to the "P\_Id" column in the "Persons" table.

The "P\_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P\_Id" 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 has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P\_Id" column when the "Orders" table is created:

CREATE TABLE Orders

(

O\_Id int NOT NULL,

OrderNo int NOT NULL,

P\_Id int,

PRIMARY KEY (O\_Id),

FOREIGN KEY (P\_Id) REFERENCES Persons(P\_Id)

)

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

SELECT MAX(Salary) FROM Employee

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

The SQL above first finds the highest salary value in the Employee table using “(select MAX(Salary) from Employee)”. Then, adding the “WHERE Salary NOT IN” in front basically creates a new set of Salary values that does not include the highest Salary value. For instance, if the highest salary in the Employee table is 200,000 then that value will be excluded from the results using the “NOT IN” operator, and all values except for 200,000 will be retained in the results.This now means that the highest value in this new result set will actually be the 2nd highest value in the Employee table. So, we then select the max Salary from the new result set, and that gives us 2nd highest Salary in the Employee table. And that is how the query above works.

**Tables for employee management system**

Emp

Empid --- primary key

CREATE TABLE Employee

(

E\_Id int NOT NULL,

LastName varchar(50) NOT NULL,

FirstName varchar(50),

Address varchar(50),

City varchar(50),

PRIMARY KEY (E\_Id)

)

empAdress

Empadressid -- primary key

Empid --- foreign key

CREATE TABLE EmployeeAddress

(

EmpAdr\_Id int NOT NULL,

MobileNo int NOT NULL,

E\_Id int,

PRIMARY KEY (EmpAdr\_Id),

FOREIGN KEY (E\_Id) REFERENCES Employee(E\_Id)

)

**Tables for e-commerce management system**

Create table Suppliers( SupplierID int PRIMARY KEY,ComapnyName varchar(55), Address varchar(90), City varchar(55), Phone int, Paymentmethods varchar(80), CustomerID int);

Products (ProductID int PRIIMARY KEY, productName varchar(50), ProductType varchar(60),Discount varchar(25),ProductAvailable varchar(25), FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID));

create table Category(CategoryId int Primary Key,categoryName varchar(24),Active varchar(45));

create table Shippers(ShipperID int Primary Key, CompanyName varchar(45),Phone varchar(45));

create table Orders(OrderId int PRIMARY KEY,OrderNumber int,Price int,Discount int,Total int,ShipDate Date);

**Tables for Library management system**

Create table Book( BookID int PRIMARY KEY, Author varchar(50), BookName varchar(55), Price int, Edition int, DateofPurchase Date);

Create table Librarian( LibrarianID int PRIMARY KEY, Password varchar(20), Name varchar(50), Address varchar(55).