1. What is Database

A database is a collection of [information](http://searchsqlserver.techtarget.com/definition/information) that is organized so that it can easily be accessed, managed, and updated.

1. What is Table

In computer programming, a table is a [data structure](http://searchsqlserver.techtarget.com/definition/data-structure) used to organize information, just as it is on paper.

1. What is Column

A **column** is a set of [data](https://en.wikipedia.org/wiki/Data) values of a particular simple [type](https://en.wikipedia.org/wiki/Datatype), one for each [row](https://en.wikipedia.org/wiki/Row_(database)) of the table. The columns provide the structure according to which the rows are composed.

1. What is Row?

 a **row**—also called a [record](https://en.wikipedia.org/wiki/Record_(computer_science)) or [tuple](https://en.wikipedia.org/wiki/Tuple" \o "Tuple)—represents a single, implicitly structured [data](https://en.wikipedia.org/wiki/Data) item in a [table](https://en.wikipedia.org/wiki/Table_(database)). In simple terms, a database table can be thought of as consisting of *rows* and [columns](https://en.wikipedia.org/wiki/Column_(database)) or [fields](https://en.wikipedia.org/wiki/Field_(computer_science)).[[1]](https://en.wikipedia.org/wiki/Row_(database)#cite_note-1) Each row in a table represents a set of related data, and every row in the table has the same structure

1. Example for 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.

Ex:

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

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

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 |

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 |

1. Example for Left outer join

A LEFT OUTER JOIN is one of the [JOIN operations](http://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj29840.html#rrefsqlj29840) that allow you to specify a join clause. It preserves the unmatched rows from the first (left) table, joining them with a NULL row in the shape of the second (right) table.

**Example 1**

**-- match cities to countries in Asia**

SELECT CITIES.COUNTRY, CITIES.CITY\_NAME, REGION

FROM Countries

LEFT OUTER JOIN Cities

ON CITIES.COUNTRY\_ISO\_CODE = COUNTRIES.COUNTRY\_ISO\_CODE

WHERE REGION = 'Asia'

**-- use the synonymous syntax, LEFT JOIN, to achieve exactly**

**-- the same results as in the example above**

SELECT COUNTRIES.COUNTRY, CITIES.CITY\_NAME,REGION

FROM COUNTRIES

LEFT JOIN CITIES

ON CITIES.COUNTRY\_ISO\_CODE = COUNTRIES.COUNTRY\_ISO\_CODE

WHERE REGION = 'Asia'

1. 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.

### SQL RIGHT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column\_name=table2.column\_name;

**get all countries and corresponding cities, including**

**-- countries without any cities**

SELECT COUNTRIES.COUNTRY, CITIES.CITY\_NAME

FROM CITIES

RIGHT OUTER JOIN COUNTRIES

ON CITIES.COUNTRY\_ISO\_CODE = COUNTRIES.COUNTRY\_ISO\_CODE

**-- get all countries in Africa and corresponding cities, including**

**-- countries without any cities**

SELECT COUNTRIES.COUNTRY, CITIES.CITY\_NAME

FROM CITIES

RIGHT OUTER JOIN COUNTRIES

ON CITIES.COUNTRY\_ISO\_CODE = COUNTRIES.COUNTRY\_ISO\_CODE

WHERE Countries.region = 'Africa'

**-- use the synonymous syntax, RIGHT JOIN, to achieve exactly**

**-- the same results as in the example above**

SELECT COUNTRIES.COUNTRY, CITIES.CITY\_NAME

FROM CITIES

RIGHT JOIN COUNTRIES

ON CITIES.COUNTRY\_ISO\_CODE = COUNTRIES.COUNTRY\_ISO\_CODE

WHERE Countries.region = 'Africa'

1. Example for Max, sun, Avg

Max: The MAX function returns the maximum value in the specified column. The MAX function can also be used as the part of sub-query.

Example

— Finds the maximum value for the column

SELECT MAX(OrderQty) FROM Purchasing.PurchaseOrderDetail

— It can be used as part of subquery

— Displays all the records having maximum OrderQty

SELECT \* FROM Purchasing.PurchaseOrderDetail WHERE OrderQty =

(SELECT MAX(OrderQty) FROM Purchasing.PurchaseOrderDetail)

**SUM**

The SUM function adds up the values in a specified column. The column must be of one of **numeric** data types. The DISTINCT can be used to get the sum of only unique values.

**Example**

— Finds the total sum for the column

SELECT SUM(OrderQty) FROM Purchasing.PurchaseOrderDetail

— Finds the sum of distinct values for the column

SELECT SUM (DISTINCT(OrderQty)) FROM Purchasing.PurchaseOrderDetail

**AVG**

The AVG function returns the average of all the values in the specified column. As with SUM function, the column must be of one of **numeric** data types. The DISTINCT can be used to get the average of only unique values.

**Example**

— Finds the average for the column

SELECT AVG(OrderQty) FROM Purchasing.PurchaseOrderDetail

— Finds the average of distinct values for the column

SELECT AVG(DISTINCT(OrderQty)) FROM Purchasing.PurchaseOrderDetail

1. Example for 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 |

1. 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:

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 |

1. Example for Where condition

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 |

1. & 13. Example for Primary key and foreign key

The **primary key** consists of one or more columns whose data contained within is used to uniquely identify eachrow in the table.  You can think of the primary key as an address.  If the rows in a table were mailboxes, then the primary key would be the listing of street addresses. When a primary key is composed of multiple columns, the data from each column is used to determine whether a row is unique.

One of the best example to understand Primary key and Foreign key in a table is Employee and Department relationshipor Customer and Order relationship.Youcancreate Order and Customer table in MySQL as following to create primary and foreign keys :

CREATE TABLE Customer (cust\_id   INT NOT NULL,  
                       cust\_name VARCHAR(256),                         
                       PRIMARY KEY (cust\_id)) ENGINE=INNODB;  
  
CREATE TABLE ORDER (order\_id INT NOT NULL,  
                    amount INT NOT NULL,  
                    cust\_id INT,  
                    FOREIGN KEY (cust\_id) REFERENCES Customer(cust\_id)  
                    ON DELETE CASCADE) ENGINE=INNODB;

Now cust\_id is primary key in Customer table and foreign key in Order table. If we try to insert an Order for which cust\_id is something which is invalid in Customer table, MySQL database will reject such INSERT or UPDATE. This is one of the benefit of using Referential Integrity. It also allow to CASCADE UPDATE and DELETE operation which first delete or update a row in parent table e.g. Customer and then delete or update all matching rows in child table e.g. Order table.

That's all on what is foreign key in a table and difference between primary and foreign key in SQL. I suggest to create some table by yourself and try to test foreign key constraint by violating it and see how database e.g. Oracle, MySQL or SQL Server  behaves. To understand more try ON DELETE CASCADE and ON DELETE UPDATE to see how database maintains foreign key constraint. You can also see my post on [Referential Integrity example on MySQL database](http://javarevisited.blogspot.sg/2012/12/what-is-referential-integrity-in-database-sql-mysql-example-tutorial.html)

In order to be a primary key, several conditions must hold true.  First, as we mentioned, the columns must be unique.  To clarify, we’re referring to the data within the rows, not the column names themselves.  Also, no value in the columns can be blank or [NULL](http://www.essentialsql.com/get-ready-to-learn-sql-server-what-is-a-null-value/).