Database SQL Important

----------LINK QUERY-------------------------------------------------

select \* from [db-srv9].[mis\_db].[dbo].claims

where claim\_no in (544659596)

------------------To Pull Log from Local/ISB servers-------------------

sp\_helptrigger Claims

--trigger name : Claims\_Updated\_trigger

--log\_db --table name : Claims\_updated

drop table #A

select \* into #A  from log\_db..Claims\_updated

where claim\_no in (544659596)

select \* from #A

order by event desc

------------------To Pull Log from NJ servers------------------------------------

https://www.codeproject.com/Articles/25600/Triggers-SQL-Server (website for trigger reading)

sp\_helptrigger Claims

--Claims\_Update\_DBTeam trigger name

--log\_db--table Name Claims\_Updated\_DBTeam

Sp\_helptexttrigger(to check script of any table)

drop table #A

select \* into #A  from log\_db..Claims\_Updated\_DBTeam

where claim\_no in (544659596)

select \* from #A

order by event desc

-------------------CLNJDB7(10.10.30.76), DB-SRV3-Cloud(10.10.30.115)-------------------------------

sp\_helptrigger Claims

--trigger name : Claims\_Updated\_trigger

--log\_db --table name : Claims\_updated

drop table #A

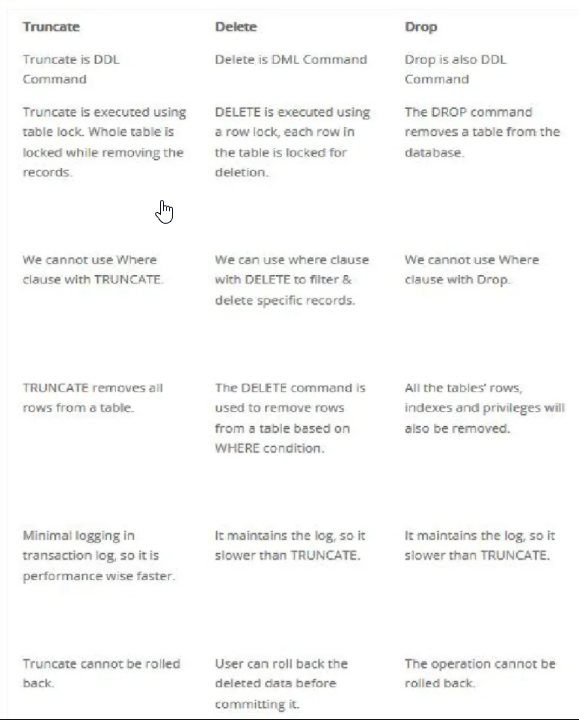
select \* into #A  from log\_db..Claims\_updated

where claim\_no in (544659596)

select \* from #A

order by event desc

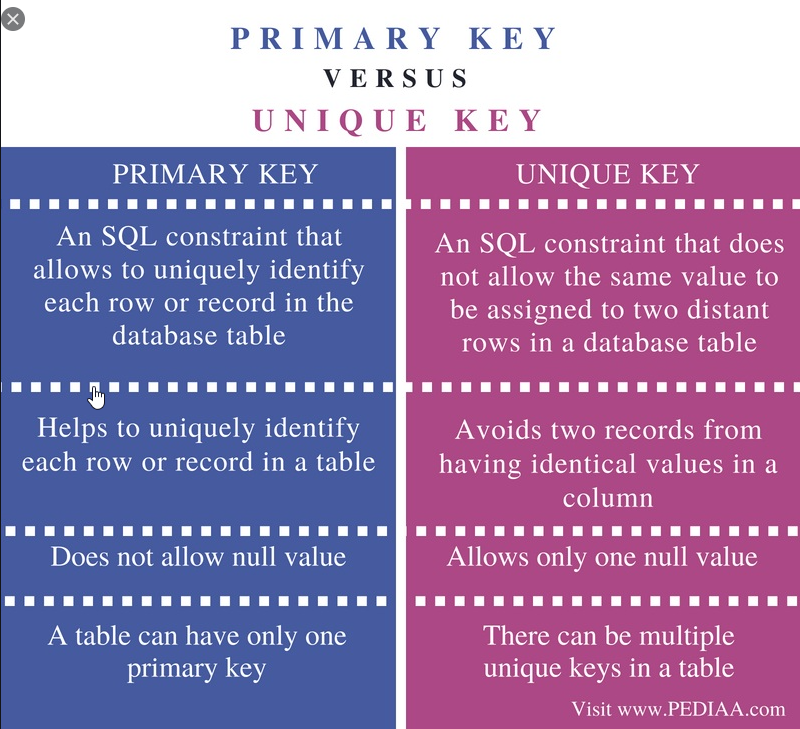
Difference between drop, truncate and delete (1/11/2020)



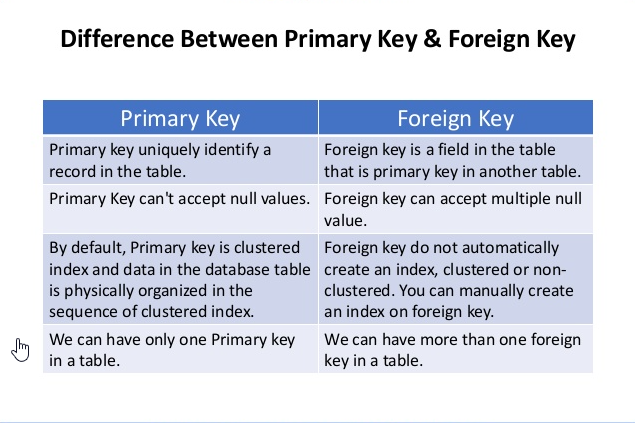
It can be rollback in sql.

But not in oracle

Difference between primary and unique key



Difference between primary and foreign key



Creating Database, Create Table, Inserting, Deleting,Truncate(Example)

use TestDB7658(for using DB through query)

go

create database persons

sp\_rename 'Persons', 'Personal';  ---to rename table name in sql

Table Creation:

CREATE TABLE Persons (

    PersonID int not null,

    LastName varchar(255),

    FirstName varchar(255),

    Address varchar(255),

    City varchar(255),

    Mobile bigint ,

    DOB date ,

    Email\_address varchar(255)

);

select \* from Persons

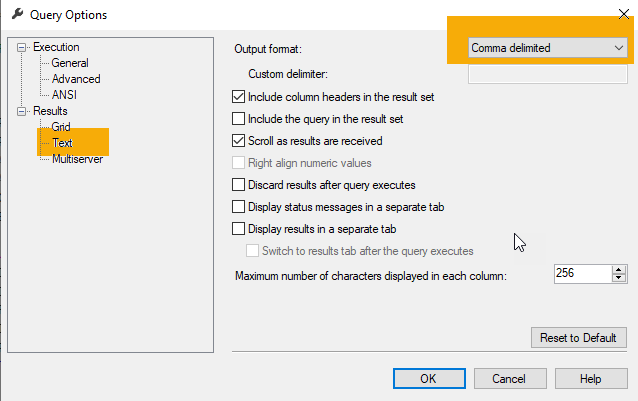
----Msg 208, Level 16, State 1, Line 15

----Invalid object name 'Persons'.

To select all columns headers at once click first write:

select top 0 \* from Persons

then click on query option on top of sql and then select below option and click ok



--PersonID,LastName,FirstName,Address,City,Mobile,DOB,Email\_address

Note: Always use begin Tran and commit\Rollback function before insert, del, update

* Insert Record:

begin Tran

insert into Persons

(PersonID,LastName,FirstName,Address,City,Mobile,DOB,Email\_address)

values ('1','Walli','Nasir','Koraal ISB','ISB',03346721665,'09/07/1984','pitafi59@gmail.com')

rollback

commit

------Msg 515, Level 16, State 2, Line 24

------Cannot insert the value NULL into column 'PersonID', table 'TestDB7658.dbo.Persons'; column does not allow nulls. INSERT fails.

------The statement has been terminated.

------Completion time: 2021-01-11T02:24:57.7401496+05:00

* UPDATE:

begin tran

update Persons

set City='Islamabad'

where PersonID='1'

rollback

commit

DROP TABLE:

drop table Persons

go

TRUNCATE:

truncate table Persons

begin tran

delete from Persons

where PersonID='1'

rollback

commit

begin tran

truncate table Persons

begin tran

drop table Persons

rollback

commit

ALTER:

ALTER TABLE Persons

ADD Father\_name varchar(255);

UPDATE:

begin tran

update Persons

set Last\_degree='BSCS'

where PersonID='1'

rollback

commit

Difference between identity and primary key

Major Difference between Primary and Identity Column

Primary Column:

* Primary Key cannot have duplicate values.
* It creates a clustered index for the Table.
* It can be set for any column type.
* We need to provide the primary column value while inserting in the table.

Identity Column:

* Identity Column can have duplicate value.
* It can only be set for Integer related columns like int, bigint, smallint, tinyint or decimal
* No need to insert values in the identity column. It is inserted automatically based on the seed.

Primary key emphasizing on uniqueness and avoid duplication value for all records on the same column, while identity provides increasing numbers in a column without inserting data. Both features could be on a single column or on difference one.

SET IDENTITY\_INSERT Persons ON---

DBCC CHECKIDENT ('Persons', RESEED, 0)---To set identity()

NOTE: If set identity on\off was not working in one session then open new session and try there its working fine.

Inserting record between two tables:

insert into Persons

select \* from Persons\_bkp\_12012021

------Msg 515, Level 16, State 2, Line 24

------Cannot insert the value NULL into column 'PersonID', table 'TestDB7658.dbo.Persons'; column does not allow nulls. INSERT fails.

------The statement has been terminated.

INTEGRITY CONSTRAINTS

Integrity Constraints

* Integrity constraints are a set of rules. It is used to maintain the quality of information.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against accidental damage to the database.

## **Types of Integrity Constraint**

### **Domain constraints**

* Domain constraints can be defined as the definition of a valid set of values for an attribute.
* The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

### **Entity integrity constraints**

* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation(table) and if the primary key has a null value, then we can't identify those rows.
* A table can contain a null value other than the primary key field.

### **Referential Integrity Constraints**

* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

### **Key constraints**

* Keys are the entity set that is used to identify an entity within its entity set uniquely.
* An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Understand SQL constraints

* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

NOTE: Constraints can be drop from table by using

Command: ALTER TABLE (Table name) DROP CONSTRAINT(constraint name)

DBMS Integrity Constraints

## **NOT NULL Constraint**

**NOT NULL** constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about this constraint is that it cannot be defined at table level.

### **Example using NOT NULL constraint**

CREATE TABLE Student(s\_id int NOT NULL, Name varchar(60), Age int);

## **UNIQUE Constraint**

**UNIQUE** constraint ensures that a field or column will only have unique values. A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

### **Using UNIQUE constraint when creating a Table (Table Level)**

Here we have a simple CREATE query to create a table, which will have a column **s\_id** with unique values.

CREATE TABLE Student(s\_id int NOT NULL UNIQUE, Name varchar(60), Age int);

## **Primary Key Constraint**

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

### **Using PRIMARY KEY constraint at Table Level**

CREATE table Student (s\_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);

The above command will creates a PRIMARY KEY on the s\_id.

### **Using PRIMARY KEY constraint at Column Level**

ALTER table Student ADD PRIMARY KEY (s\_id);

## **Foreign Key Constraint**

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables.

## **CHECK Constraint**

**CHECK** constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database.

\

# [Difference Between Local and Global Temporary Tables in SQL Server](https://database.guide/difference-between-local-and-global-temporary-tables-in-sql-server/)

## **Naming**

* Local temporary tables’ names start with a single number sign (#). For example, #My Table.
* Local temporary tables also get a system generated numeric suffix appended to the name. This is automatically generated by SQL Server. This allows multiple sessions to create local temporary tables with the same name without causing naming conflicts. However, users can still query the table without having to know the suffix.
* Global temporary tables’ names start with a double number sign (##). For example, ##My Table.
* Global temporary tables are not appended with a suffix like with local temporary tables.

## **Visibility**

* Local temporary tables are visible only in the current session.
* Global temporary tables are visible to all sessions.

## **Lifespan**

* Local temporary tables are automatically dropped at the end of the current session. A local temporary table created in a stored procedure is dropped automatically when the stored procedure is finished. The table can be referenced by any nested stored procedures executed by the stored procedure that created the table. The table cannot be referenced by the process that called the stored procedure that created the table.
* Global temporary tables are automatically dropped when the session that created the table ends and all other tasks have stopped referencing them. The association between a task and a table is maintained only for the life of a single T-SQL statement. Therefore, a global temporary table is dropped at the completion of the last T-SQL statement that was actively referencing the table when the creating session ended.

# GROUP BY and HAVING Clause in SQL

## **Group By Clause**

The GROUP BY Clause is utilized in SQL with the SELECT statement to organize similar data into groups. It combines the multiple records in single or more columns using some functions. Generally, these functions are aggregate functions such as min(),max(),avg(), count(), and sum() to combine into single or multiple columns. It uses the **split-apply-combine** strategy for data analysis.

* In the split phase, It divides the groups with its values.
* In the apply phase, It applies the aggregate function and generates a single value.
* In the combiner phase, It combines the groups with single values into a single value.

**Points to Remember:**

* GROUP BY Clause is utilized with the SELECT statement.
* GROUP BY aggregates the results on the basis of selected column: COUNT, MAX, MIN, SUM, AVG, etc.
* GROUP BY returns only one result per group of data.
* GROUP BY Clause always follows the WHERE Clause.
* GROUP BY Clause always precedes the ORDER BY

## **Having Clause**

HAVING Clause utilized in SQL as a conditional Clause with GROUP BY Clause. This conditional clause returns rows where aggregate function results matched with given conditions only. It added in the SQL because WHERE Clause cannot be combined with aggregate results, so it has a different purpose. The primary purpose of the WHERE Clause is to deal with non-aggregated or individual records.

* HAVING Clause always utilized in combination with GROUP BY Clause.
* HAVING Clause restricts the data on the group records rather than individual records.
* WHERE and HAVING can be used in a single query.

# What is the difference between UNION and UNION ALL

The main difference between UNION and UNION ALL is that:

* **UNION:** only keeps *unique* records
* **UNION ALL:** keeps all records, including *duplicates*

### **UNION:**

UNION removes any duplicate records. UNION first performs a sorting operation and eliminates of the records that are duplicated across all columns before finally returning the combined data set.

### **UNION ALL**

If we were to now perform the UNION ALL on the same data set, the query would skip the deduplication step and return the results shown.

### **OFF SET FETCH:**

OFFSET and FETCH Clause are used in conjunction with SELECT and ORDER BY clause to provide a means to retrieve a range of records.

**OFFSET**

The OFFSET argument is used to identify the starting point to return rows from a result set. Basically, it exclude the first set of records.  
**Note:**

* OFFSET can only be used with ORDER BY clause. It cannot be used on its own.
* OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

Syntax:

SELECT column name(s)

FROM table name

WHERE condition

ORDER BY column name

OFFSET rows\_to\_skip ROWS;

### **SUB-QUERY:**

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

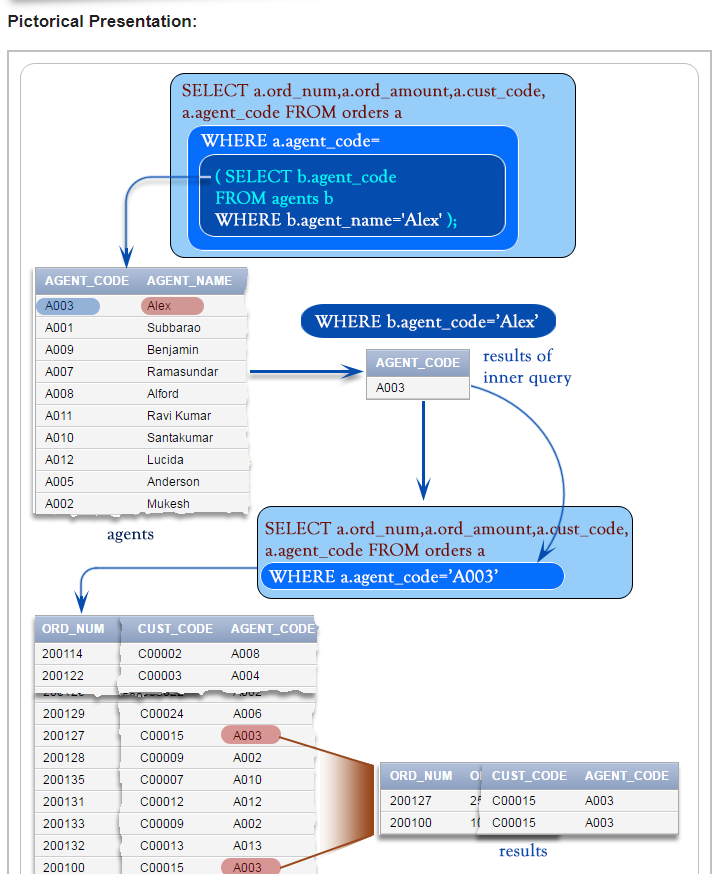
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

### **There are a few rules that subqueries must follow −**

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators such as the iN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.
* You can use upto 32 sub queries in one script.(In SQL server)

### **Correlated Subqueries**

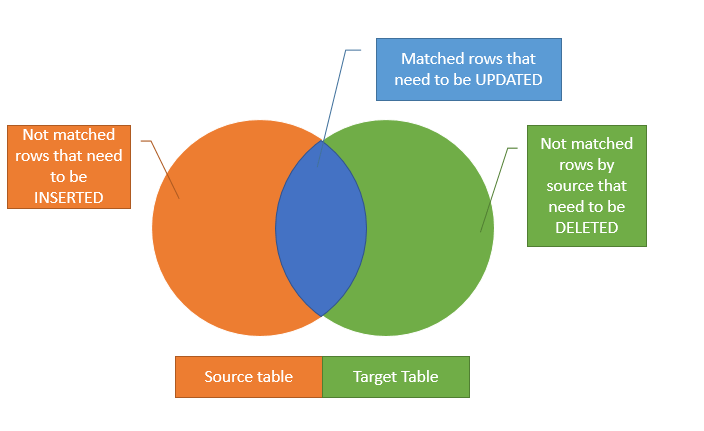
SQL Correlated Subqueries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query. In this type of queries, a table alias (also called a correlation name) must be used to specify which table reference is to be used.



### **SQL Server MERGE Statement**

Suppose, you have two table called source and target tables, and you need to update the target table based on the values matched from the source table. There are three cases:

1. The source table has some rows that do not exist in the target table. In this case, you need to [insert](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/) rows that are in the source table into the target table.
2. The target table has some rows that do not exist in the source table. In this case, you need to [delete](https://www.sqlservertutorial.net/sql-server-basics/sql-server-delete/) rows from the target table.
3. The source table has some rows with the same keys as the rows in the target table. However, these rows have different values in the non-key columns. In this case, you need to [update](https://www.sqlservertutorial.net/sql-server-basics/sql-server-update/) the rows in the target table with the values coming from the source table.



SYNTAX:

**MERGE** target\_table **USING** source\_table

**ON** merge\_condition

**WHEN** **MATCHED**

**THEN** update\_statement

**WHEN** **NOT** **MATCHED**

**THEN** insert\_statement

**WHEN** **NOT** **MATCHED** **BY** **SOURCE**

**THEN** **DELETE**;

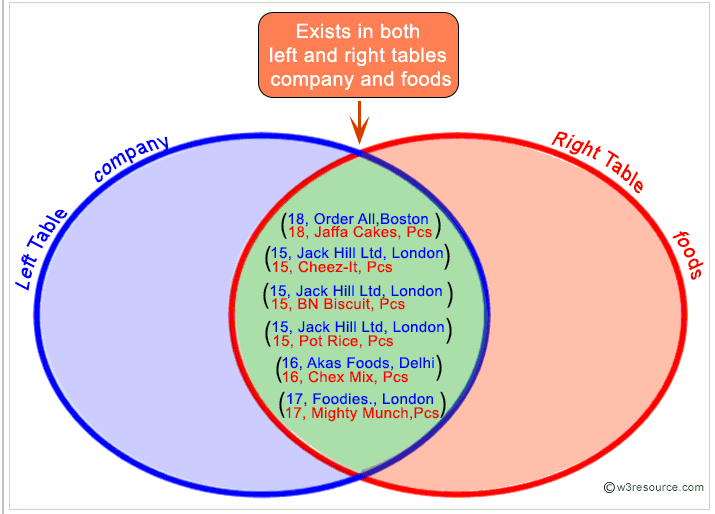
# ****JOIN:****

# JOIN returns all rows from tables where the key record of one table is equal to the key records of another table.

# **SQL INNER JOIN:**

# The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns. An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

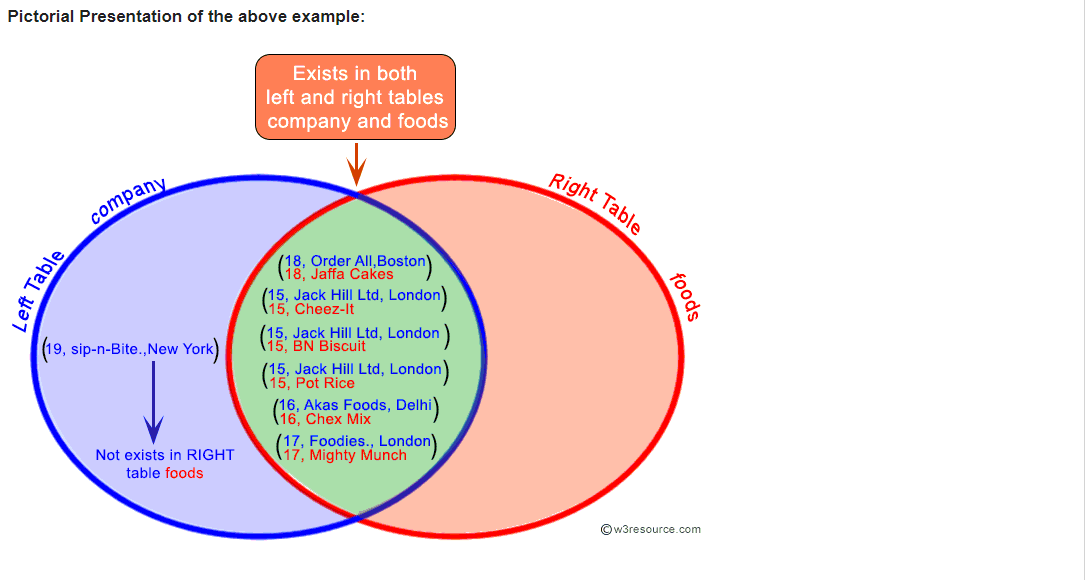
# **Pictorial Presentation:**



# ****LEFT JOIN:****

# The SQL LEFT JOIN (specified with the keywords LEFT JOIN and ON) joins two tables and fetches all matching rows of two tables for which the SQL-expression is true, plus rows from the first table (left ) that do not match any row in the second table.

LEFT JOIN Diagram:



# **RIGHT JOIN**

# The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which is matching in both the tables (before and after the JOIN clause mentioned in the syntax below) , and the unmatched rows will also be available from the table written after the JOIN clause ( mentioned in the syntax below ).

**Syntax:**

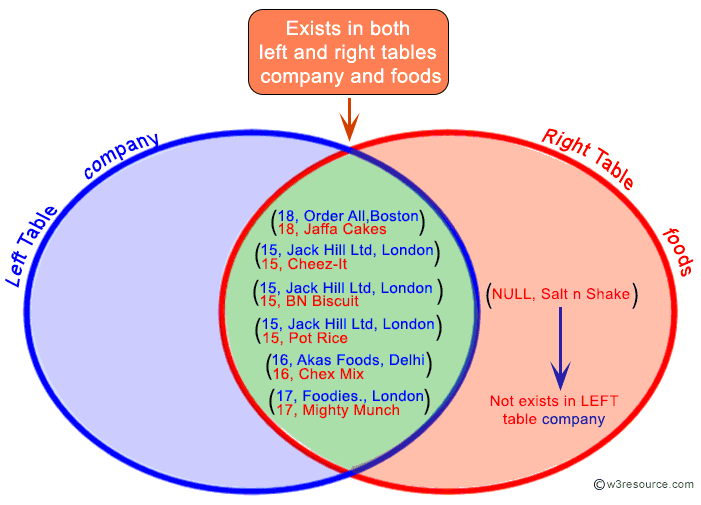
SELECT \*

FROM table1

RIGHT [ OUTER ] JOIN table2

ON table1.column\_name=table2.column\_name;

Pictorial Presentation:



# ****Full Outer Join:****

# In SQL the FULL OUTER JOIN combines the results of both [left](https://www.w3resource.com/sql/joins/perform-a-left-join.php)and [right](https://www.w3resource.com/sql/joins/perform-a-right-join.php) outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.

**Syntax:**

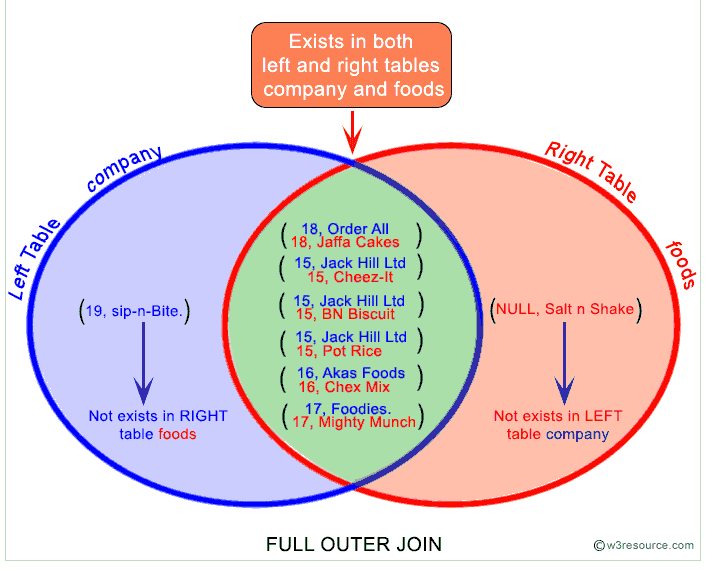
SELECT \*

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name=table2.column\_name;

Pictorial Presentation:



# **Self Join:**

# A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

# Syntax:

SELECT a.column\_name, b.column\_name...

FROM table1 a, table1 b

WHERE a.common\_field = b.common\_field;

# **SQL Cross Join:**

The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian Product.

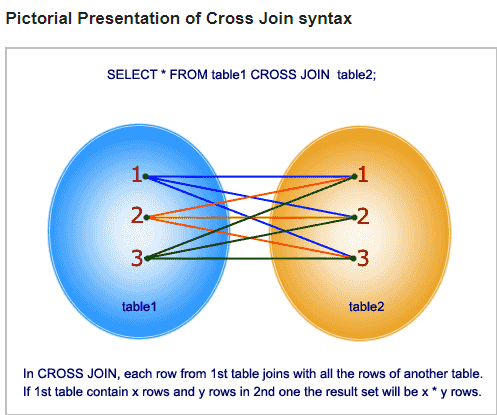
If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

**Syntax:**

SELECT \*

FROM table1

CROSS JOIN table2;



# **SQL Equi Join:**

SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.

You may also perform EQUI JOIN by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

**Syntax:**

SELECT column\_list

FROM table1, table2....

WHERE table1.column\_name =

table2.column\_name;

or

SELECT \*

FROM table1

JOIN table2

[ON (join\_condition)]

# 

The difference here is for the Inner join i can have a different condition , such as “=” , <> , > , < etc, but same is not true for equi join.

Notes :

1. Inner join can have equality (=) and other operators (like <,>,<>) in the join condition.
2. Equi join only have equality (=) operator in the join condition.
3. Equi join can be an Inner join, Left Outer join, Right Outer join
4. The USING clause is not supported by SQL Server and Sybase. This clause is supported by Oracle and MySQL.

# ****NON EQUI JOIN****

The SQL NON EQUI JOIN uses comparison operator instead of the equal sign like **>, <, >=, <=**along with conditions.

**Syntax:**

SELECT \*

FROM table\_name1, table\_name2

WHERE table\_name1.column [> | < | >= | <= ] table\_name2.column;

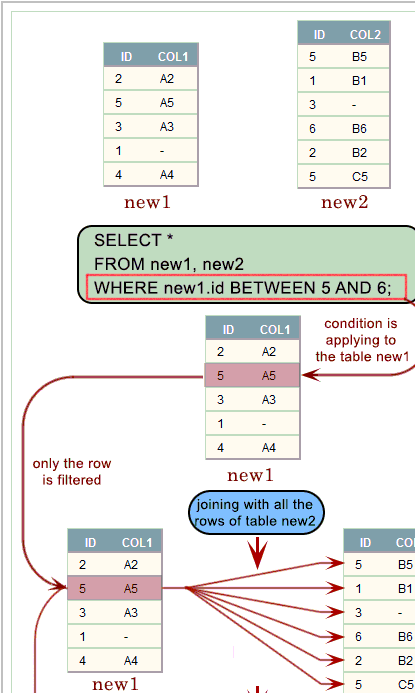
**Theta Join:**

**THETA JOIN** allows you to merge two tables based on the condition represented by **theta**. **Theta joins** work for all comparison operators. It is denoted by symbol θ. The general case of **JOIN** operation is called a **Theta join**

**THETA JOIN** allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators. It is denoted by symbol **θ**. The general case of JOIN operation is called a Theta join.

Syntax:

A ⋈θ B



# **Natural Join**

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associated tables will appear once only.

**Natural Join: Guidelines**

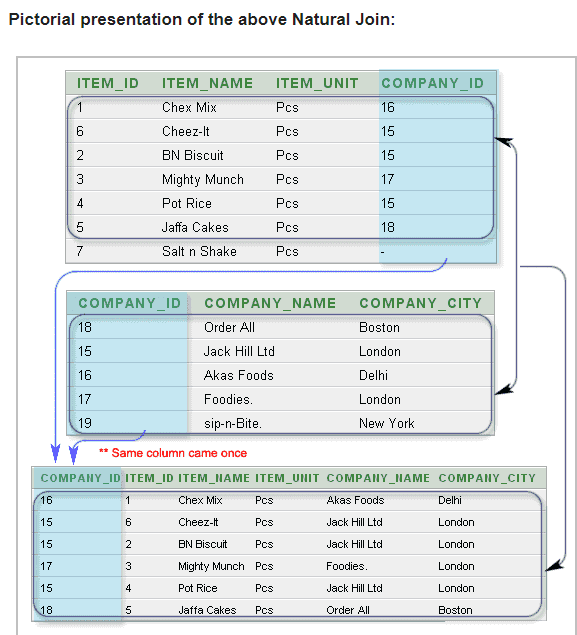
- The associated tables have one or more pairs of identically named columns.  
- The columns must be the same data type.  
- Don’t use ON clause in a natural join.

**Syntax:**

SELECT \*

FROM table1

NATURAL JOIN table2;



**CROSS APPLY and OUTER APPLY**

* The CROSS APPLY operator returns only those rows from the left table expression (in its final output) if it matches with the right table expression. In other words, the right table expression returns rows for the left table expression match only.
* The OUTER APPLY operator returns all the rows from the left table expression irrespective of its match with the right table expression. For those rows for which there are no corresponding matches in the right table expression, it contains NULL values in columns of the right table expression.
* So you might conclude, the CROSS APPLY is equivalent to an INNER JOIN (or to be more precise its like a CROSS JOIN with a correlated sub-query) with an implicit join condition of 1=1 whereas the OUTER APPLY is equivalent to a LEFT OUTER JOIN.

**WHY:**

when do you use the APPLY operator? Although the same can be achieved with a [normal JOIN](https://www.mssqltips.com/sqlservertip/1667/sql-server-join-example/), the need of APPLY arises if you have a table-valued expression on the right part and in some cases the use of the APPLY operator boosts [performance](https://www.mssqltips.com/sql-server-tip-category/9/performance-tuning/) of your query.

# EXISTS Condition:

* The SQL Server (Transact-SQL) EXISTS condition is used in combination with a subquery and is considered to be met if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

### **Parameters or Arguments**

The subquery is a SELECT statement. If the subquery returns at least one record in its result set, the EXISTS clause will evaluate to true and the EXISTS condition will be met. If the subquery does not return any records, the EXISTS clause will evaluate to false and the EXISTS condition will not be met.

## Note

* SQL statements that use the EXISTS condition are very inefficient since the sub-query is RE-RUN for EVERY row in the outer query's table. There are more efficient ways to write most queries, that do not use the EXISTS condition.

## Example

SELECT \*

FROM employees

WHERE EXISTS (SELECT \*

FROM contacts

WHERE employees.last\_name = contacts.last\_name

AND employees.first\_name = contacts.first\_name);

**Result:**

This SQL Server EXISTS condition example will return all records from the employees table where there is at least one record in the contacts table with a matching last\_name and first\_name.

## NOT EXISTS

The SQL Server EXISTS condition can also be combined with the[NOT operator](https://www.techonthenet.com/sql_server/not.php).