SQL

Topics

- Where Clause
  - Types Of Conditions
  - Comparison
  - BETWEEN
  - EXPRESSION
  - IN
  - NULL
  - LIKE
- Aggregate Function
- Ordered By
- Group By
- Having
- OFFSET AND FETCH
- SELECT WITH TOP AND DISTINCT
- ALIAS
- Types of Joins
  - where Clause
  - Group by
  - Having
- Subquery
- UNION vs UNION ALL
- TEMP TABLE
- CTE RECURSIVE

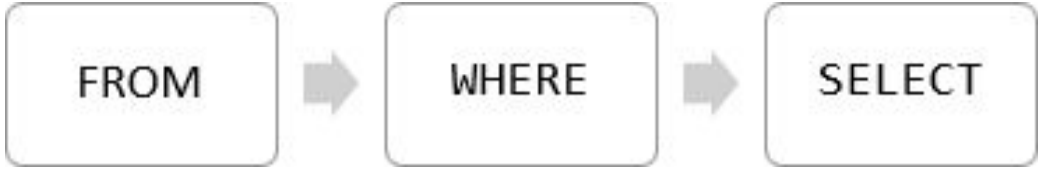
Where Clause

- The basic form of the SELECT statement is SELECT-FROM-WHERE block. In a SELECT statement, WHERE clause is optional. Using SELECT without a WHERE clause is useful for browsing data from tables.
- In a WHERE clause, you can specify a search condition (logical expression) that has one or more conditions. When the condition (logical expression) evaluates to true the WHERE clause filter unwanted rows from the result.

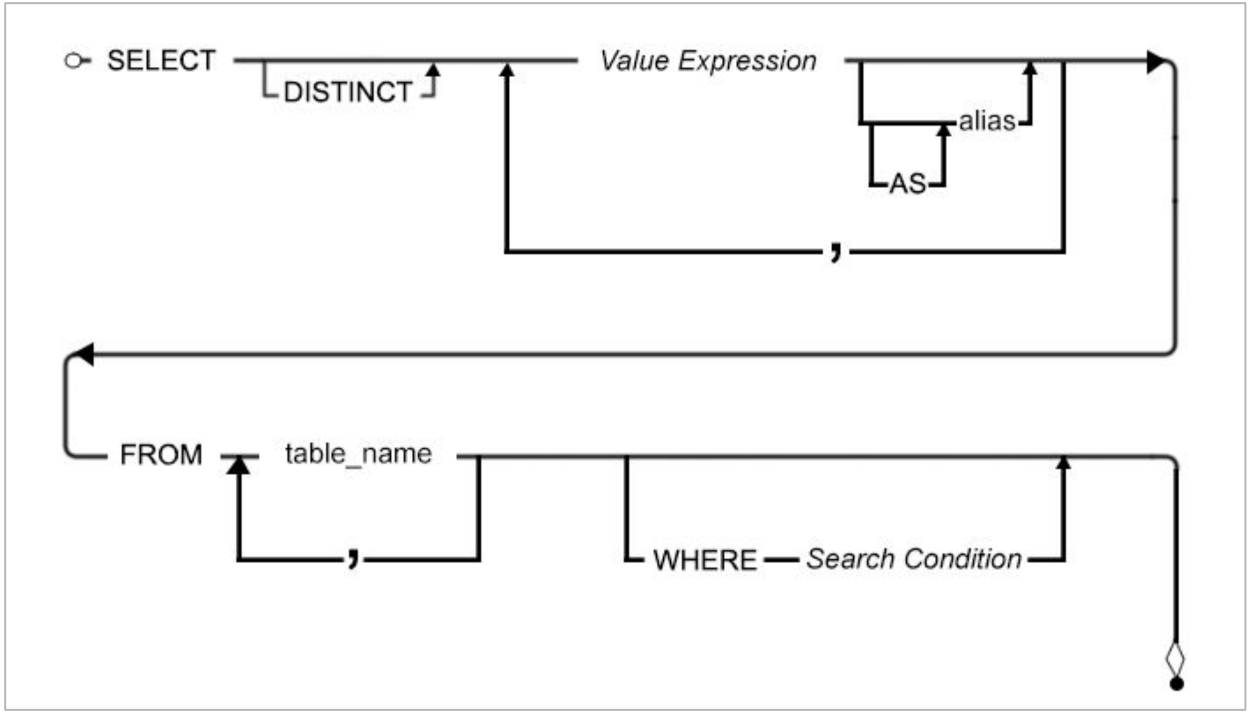
Syntax:-

```
SELECT <column_list>
FROM < table name >
  WHERE <condition>;
```

- When the WHERE clause is available, SQL Server processes the clauses of the query in the following sequence: FROM, WHERE, and SELECT.



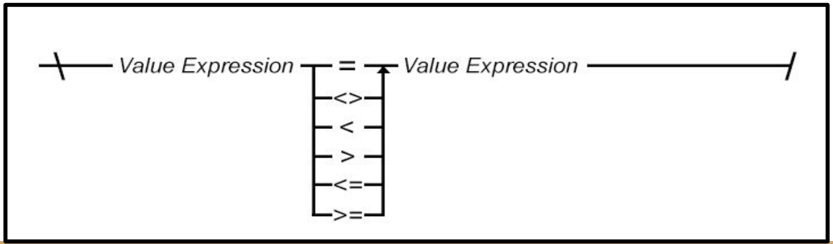
Syntax Diagram



Types Of Conditions

Condition	**SQL Operators**
Comparison	=, >, >=, <, <=, <>
Range filtering	BETWEEN
Match a character pattern	LIKE
List filtering [Match any of a list of values]	IN
Null testing	IS NULL

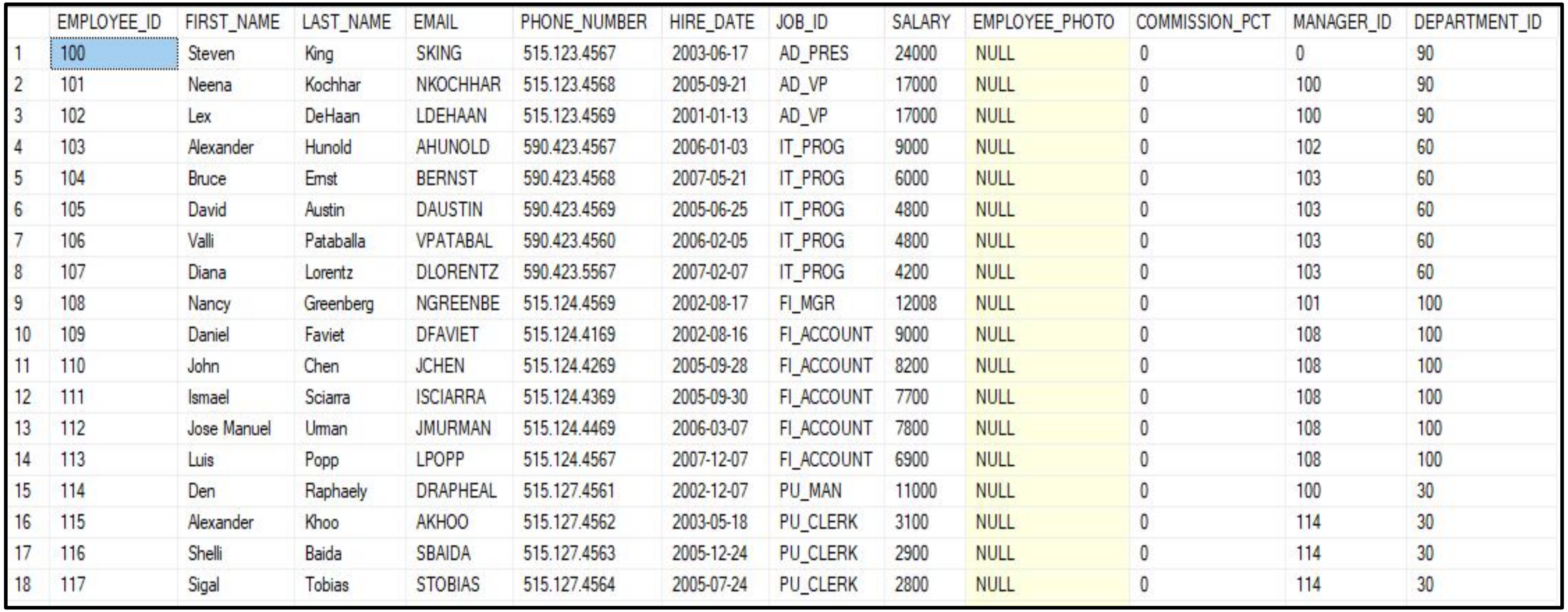
Comparison condition - Syntax diagram



Comparison Operators

SQL Operators	**Meaning**
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<>	Not equal to
<=	Less than or equal to

Sample Table



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	EMPLOYEE_PHOTO	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	2003-06-17	AD_PRES	24000	NULL	0	0	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21	AD_VP	17000	NULL	0	100	90
3	102	Lex	DeHaan	LDEHAAN	515.123.4569	2001-01-13	AD_VP	17000	NULL	0	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03	IT_PROG	9000	NULL	0	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21	IT_PROG	6000	NULL	0	103	60
6	105	David	Austin	DAUSTIN	590.423.4569	2005-06-25	IT_PROG	4800	NULL	0	103	60
7	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05	IT_PROG	4800	NULL	0	103	60
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07	IT_PROG	4200	NULL	0	103	60
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-17	FI_MGR	12008	NULL	0	101	100
10	109	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-16	FI_ACCOUNT	9000	NULL	0	108	100
11	110	John	Chen	JCHEN	515.124.4269	2005-09-28	FI_ACCOUNT	8200	NULL	0	108	100
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30	FI_ACCOUNT	7700	NULL	0	108	100
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-07	FI_ACCOUNT	7800	NULL	0	108	100
14	113	Luis	Popp	LPOPP	515.124.4567	2007-12-07	FI_ACCOUNT	6900	NULL	0	108	100
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-07	PU_MAN	11000	NULL	0	100	30
16	115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-18	PU_CLERK	3100	NULL	0	114	30
17	116	Shelli	Baida	SBAIDA	515.127.4563	2005-12-24	PU_CLERK	2900	NULL	0	114	30
18	117	Sigal	Tobias	STOBIAS	515.127.4564	2005-07-24	PU_CLERK	2800	NULL	0	114	30

Example

Problem Statement:-Display the employee_id, first_name, last_name, department_id of employees whose departmet_id=100;

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id = 100;
```

WHERE clause using comparison conditions

Problem Statement:-Displays the employee_id, first_name, last_name and salary of employees whose salary is greater than or equal to 4000.

```
SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE salary >= 4000;
```

WHERE clause using BETWEEN condition

The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression. The operator returns TRUE when the search value present within the range otherwise returns FALSE. The results are NULL if any of the range values are NULL.

Example:-

Problem Statement:-Displays the employee_id, first_name, last_name and salary of employees whose salary is greater than or equal to 4000 and less than equal to 6000 where 4000 is the lower limit and 6000 is the upper limit of the salary.

```
SELECT employee_id, first_name, last_name, salary
FROM employees
-- Filtering the results to include only rows where the 'salary' is between 4000 and 6000
WHERE salary BETWEEN 4000 AND 6000;
```

WHERE clause using expression

Problem Statement:- Displays the first_name, last_name , salary and (salary+(salary*commission_pct)) as Net Salary of employees whose Net Salary is in the range 10000 and 15000 and who gets at least a percentage of commission_pct.

```
SELECT first_name, last_name, salary, (salary + (salary * commission_pct))
"Net Salary"
FROM employees
-- Filtering the results to include only rows where the "Net Salary" is between 10000 and
15000
-- and the commission_pct is greater than 0
WHERE (salary + (salary * commission_pct)) BETWEEN 10000 AND 15000 AND
commission_pct > 0;
```

WHERE clause using IN condition

- The IN operator checks a value within a set of values separated by commas and retrieve the rows from the table which are matching. The IN returns 1 when the search value present within the range otherwise returns 0.

Example:-

Problem Statement:-Displays the employee_id, first_name, last_name, department_id and salary of employees whose department_id 60, 90 or 100.

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
-- Filtering the results to include only rows where the 'department_id' is in the list (60,
90, 100)
WHERE department_id IN (60, 90, 100);
```

WHERE clause using NULL condition

- SQL Null check is performed using either IS NULL or IS NOT NULL to check whether a value in a field is NULL or not.
- When a field value is NULL it means that the database assigned nothing in that field for that row. The NULL is not zero or blank. It represents an unknown or inapplicable value. It can't be compared using AND / OR logical operators. The special operator 'IS' is used with the keyword 'NULL' to locate 'NULL' values. NULL can be assigned in both type of fields i.e. numeric or character type of field.

Example:-

Problem Statement:-Displays the employee_id, first_name, last_name and salary of employees whose department_id is null.

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
-- Filtering the results to include only rows where the 'department_id' is NULL
WHERE department_id IS NULL;
```

WHERE clause using LIKE condition

- The SQL Server LIKE is a logical operator that determines if a character string matches a specified pattern. A pattern may include regular characters and wildcard characters. The LIKE operator is used in the WHERE clause of the SELECT, UPDATE, and DELETE statements to filter rows based on pattern matching.
- The following illustrates the syntax of the SQL Server LIKE operator:

```
column | expression LIKE pattern [ESCAPE escape_character]
```

Pattern

- The pattern is a sequence of characters to search for in the column or expression. It can include the following valid wildcard characters:
  - The percent wildcard (%): any string of zero or more characters.
  - The underscore (_) wildcard: any single character.
  - The [list of characters] wildcard: any single character within the specified set.
  - The [character-character]: any single character within the specified range.
  - The [^]: any single character not within a list or a range.
- The wildcard characters makes the LIKE operator more flexible than the equal (=) and not equal (!=) string comparison operators.

Example

Problem statement 1 :-Displays the employee_id, first_name, last_name and salary of employees whose first_name starting with 'S'.

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
-- Filtering the results to include only rows where the 'first_name' starts with the letter 'S'
WHERE first_name LIKE ('S%');
```

Problem Statement 2:-Displays the employee_id, first_name, last_name and salary of employees whose first_name is not the letter in the range A through X

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
-- Filtering the results to include only rows where the 'first_name' is not in range A through X
WHERE first_name LIKE '[^A-X]%';
```

Aggregate Function

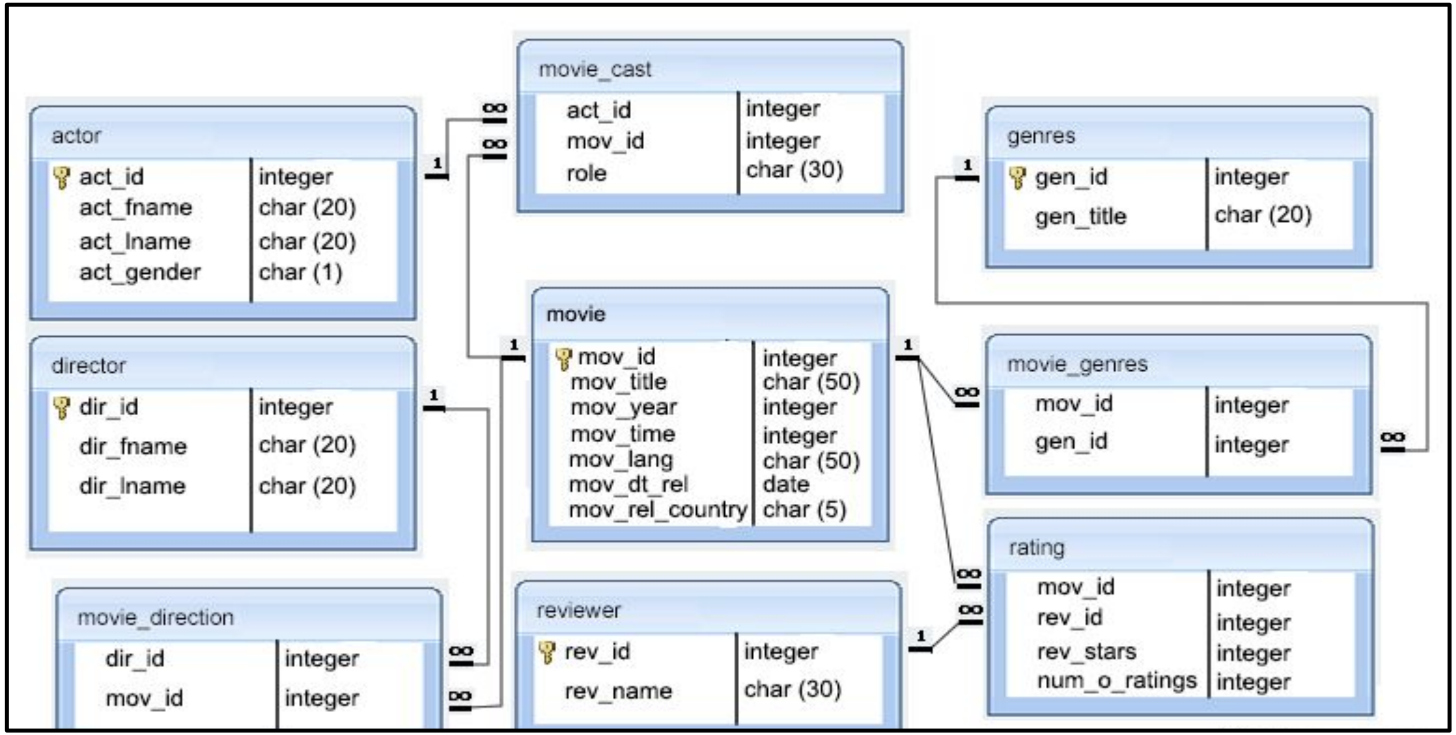
- Each query in SQL returns filtered results of groups of values and also the field values. SQL provides aggregate functions to help with the summarization of large volumes of data.
- This function can produce a single value for an entire group or table.
- They operate on sets of rows and return results based on groups of rows.
- The general syntax for most of the aggregate function is as follows:

```
aggregate_function( [ DISTINCT | ALL ] expression)
```

List of SQL Aggregate functions are

Functions	**Description**
SQL Count function	The SQL COUNT function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. It sets the number of rows or non NULL column values.
SQL Sum function	The SQL AGGREGATE SUM() function returns the sum of all selected column.
SQL Avg function	The SQL AVG function calculates the average value of a column of numeric type. It returns the average of all non NULL values
SQL Max function	The aggregate function SQL MAX() is used to find the maximum value or highest value of a certain column or expression. This function is useful to determine the largest of all selected values of a column.
SQL Min function	The aggregate function SQL MIN() is used to find the minimum value or lowest value of a column or expression. This function is useful to determine the smallest of all selected values of a column.

Sample Data


movie_cast
act_id integer
mov_id integer
role char (30)
actor
act_id integer
act_fname char (20)
act_lname char (20)
act_gender char (1)
genres
gen_id integer
gen_title char (20)
director
dir_id integer
dir_fname char (20)
dir_lname char (20)
movie
mov_id integer
mov_title char (50)
mov_year integer
mov_time integer
mov_lang char (50)
mov_dt_rel date
mov_rel_country char (5)
movie_genres
mov_id integer
gen_id integer
rating
mov_id integer
rev_id integer
rev_stars integer
num_o_ratings integer
movie_direction
dir_id integer
mov_id integer
reviewer
rev_id integer
rev_name char (30)
1
∞

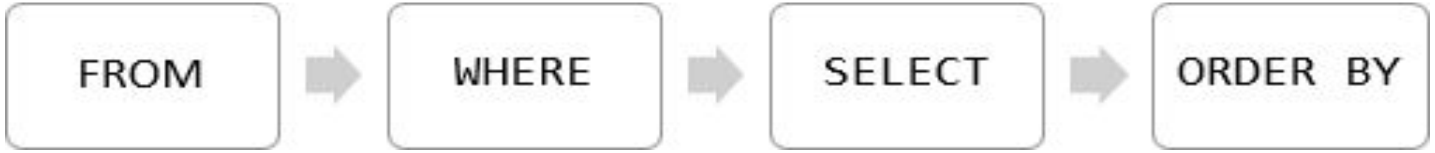
Order By

- The ORDER BY clause orders or sorts the result of a query according to the values in one or more specific columns. More than one columns can be ordered one within another. It depends on the user that, whether to order them in ascending or descending order. The default order is ascending.
- The SQL ORDER BY clause is used with the SQL SELECT statement.
- **Syntax:**

```
SELECT <column_list> FROM < table name >.
WHERE <condition>
  ORDER BY <columns> [ASC | DESC];
```

How to build Query

In this case, SQL Server processes the clauses of the query in the following sequence: FROM, WHERE, SELECT, and ORDER BY.



Problem Statement:- write a SQL query to find the name and year of the movies. Return movie title and sort the data by movie release year in ascending order.

```
SELECT mov_title, mov_year
FROM tbl_movies order by mov_year;
```

Example

Write a SQL query to find the movie titles that contain the word 'Boogie Nights'. Sort the result-set in ascending order by movie year. Return movie ID, movie title and movie release year.

```
-- Selecting 'mov_id', 'mov_title', and 'mov_year' from the 'movie' table
-- Filtering results where 'mov_title' contains the substring 'Boogie Nights'
-- Ordering the results by 'mov_year' in ascending order
SELECT mov_id, mov_title, mov_year
FROM tbl_movies
WHERE mov_title LIKE '%Boogie%Nights%'
ORDER BY mov_year ASC;
```

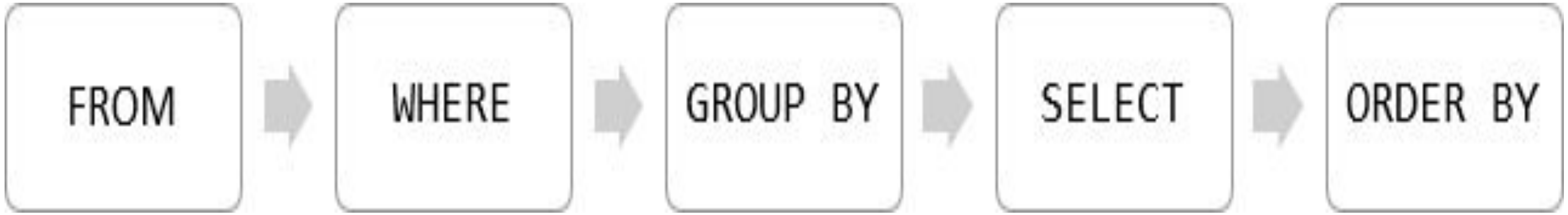
Group By

- The usage of SQL GROUP BY clause is, to divide the rows in a table into smaller groups.
- The GROUP BY clause is used with the SQL SELECT statement.
- The grouping can happen after retrieves the rows from a table.
- When some rows are retrieved from a grouped result against some condition, that is possible with HAVING clause.
- The GROUP BY clause is used with the SELECT statement to make a group of rows based on the values of a specific column or expression. The SQL AGGREGATE function can be used to get summary information for every group and these are applied to an individual group.
- The WHERE clause is used to retrieve rows based on a certain condition, but it can not be applied to grouped result.
- In an SQL statement, suppose you are using GROUP BY, if required you can use HAVING instead of WHERE, after GROUP BY.
- **Syntax:-**

```
SELECT <column_list>
FROM < table name >
WHERE <condition>GROUP BY <columns>
[HAVING] <condition>;
```

How to build Query

In this case, SQL Server processes the clauses in the following sequence: FROM, WHERE, GROUP BY, SELECT, and ORDER BY.



Example

Problem Statement: Count the Number of Movies Released Each Year

```
SELECT
    mov_year AS ReleaseYear,
    COUNT(*) AS NumberOfMovies
FROM
    movie
GROUP BY
    mov_year
ORDER BY
    mov_year;
```

Having Clause

- SQL HAVING clause specifies a search condition for a group or an aggregate. HAVING is usually used in a GROUP BY clause, but even if you are not using GROUP BY clause, you can use HAVING to function like a WHERE clause. You must use HAVING with SQL SELECT.

```
SELECT <column_list> FROM < table name >
WHERE <search_condition]>
GROUP BY <columns>
[HAVING] <search_condition]>
  [ORDER BY {order_expression [ASC | DESC]}[, ...]];
```

How a HAVING clause works IN SQL?

- The select clause specifies the columns.
- The from clause supplies a set of potential rows for the result.
- The where clause gives a filter for these potential rows.
- The group by clause divide the rows in a table into smaller groups.
- The having clause gives a filter for these group rows.

Example

Problem Statement: Find Movies with Average Rating Above a Threshold

```
SELECT mov_id AS MovieID, AVG(rev_stars) AS AverageRating
FROM
    tbl_rating
WHERE
    num_of_rating > 0 -- Exclude movies with zero ratings
GROUP BY
    mov_id
HAVING
    AVG(rev_stars) > 4.0
ORDER BY
    AverageRating DESC;
```

OFFSET AND FETCH

- The OFFSET and FETCH clauses are the options of the ORDER BY clause. They allow you to limit the number of rows to be returned by a query.
- The following illustrates the syntax of the OFFSET and FETCH clauses:

```
ORDER BY column_list [ASC |DESC]
OFFSET offset_row_count {ROW | ROWS}
FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} ONLY
```

OFFSET AND FETCH Contd.

There are some contexts that Explain

The OFFSET clause specifies the number of rows to skip before starting to return rows from the query. The `offset_row_count` can be a constant, variable, or parameter that is greater or equal to zero.

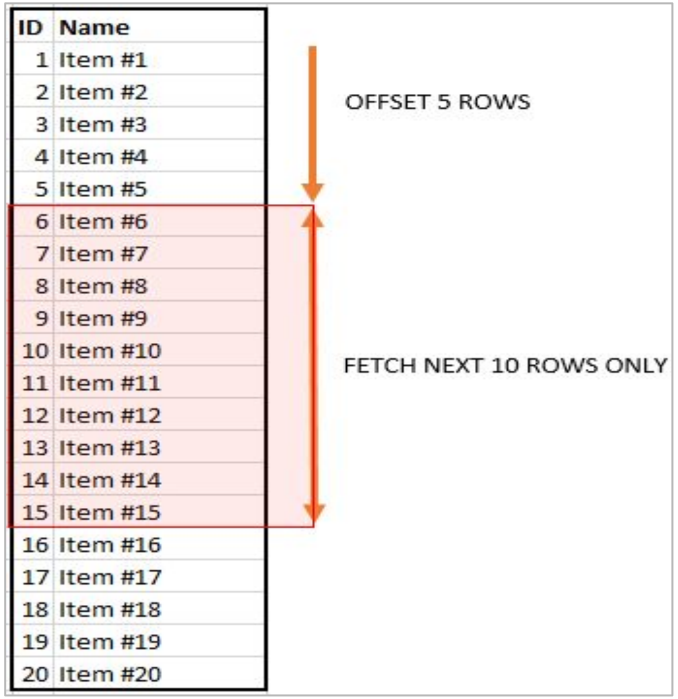
```
OFFSET offset_row_count {ROW | ROWS}
```

The FETCH clause specifies the number of rows to return after the OFFSET clause has been processed. The `offset_row_count` can a constant, variable or scalar that is greater or equal to one.

```
FETCH {FIRST | NEXT} fetch_row_count
{ROW | ROWS} ONLY
```

OFFSET AND FETCH Contd.

The following illustrates the OFFSET and FETCH clauses:



ID	Name
1	Item #1
2	Item #2
3	Item #3
4	Item #4
5	Item #5
6	Item #6
7	Item #7
8	Item #8
9	Item #9
10	Item #10
11	Item #11
12	Item #12
13	Item #13
14	Item #14
15	Item #15
16	Item #16
17	Item #17
18	Item #18
19	Item #19
20	Item #20

Note that you must use the OFFSET and FETCH clauses with the ORDER BY clause. Otherwise, you will get an error.

The OFFSET and FETCH clauses are preferable for implementing the query paging solution than the TOP clause.

Example

- Retrieve the top 10 latest movies

```
SELECT * FROM
    tbl_movies
ORDER BY
    mov_dt_rel DESC
OFFSET 0 ROWS -- Skip 0 rows (start from the beginning)
FETCH FIRST 10 ROWS ONLY; -- Fetch the top 10 rows
```

SELECT TOP

- The SELECT TOP clause allows you to limit the number of rows or percentage of rows returned in a query result set.
- Because the order of rows stored in a table is unspecified, the SELECT TOP statement is always used in conjunction with the ORDER BY clause. Therefore, the result set is limited to the first N number of ordered rows.
- The following shows the syntax of the `TOP` clause with the `SELECT` statement:

```
SELECT TOP (expression) [PERCENT]
   [WITH TIES]
FROM
   table_name
ORDER BY
   column_name;
```

SELECT TOP Contd.

expression	**Following the TOP keyword is an expression that specifies the number of rows to be returned. The expression is evaluated to a float value if PERCENT is used, otherwise, it is converted to a BIGINT value.**
PERCENT	**The PERCENT keyword indicates that the query returns the first N percentage of rows, where N is the result of the expression.**
WITH TIES	**The WITH TIES allows you to return more rows with values that match the last row in the limited result set. Note that WITH TIES may cause more rows to be returned than you specify in the expression.**

For example, if you want to return the most expensive products, you can use the TOP 1. However, if two or more products have the same prices as the most expensive product, then you miss the other most expensive products in the result set.
To avoid this, you can use TOP 1 WITH TIES. It will include not only the first expensive product but also the second one, and so on.

Example

Problem Statement:- Retrieves the top 10 movies based on their release date

```
SELECT TOP 10 with TIES mov_id, mov_title, mov_year, mov_dt_rel
FROM tbl_movies
ORDER BY
    mov_year DESC;
```

Problem Statement:- Retrieves the top-rated movies based on average reviewer stars

```
SELECT TOP 10 WITH TIES m.mov_id, m.mov_title, AVG(r.rev_stars) AS avg_rating
FROM tbl_movies m JOIN tbl_rating r ON m.mov_id = r.mov_id
GROUP BY m.mov_id, m.mov_title
ORDER BY avg_rating DESC;
```

Select with distinct

- Redundancy is the repetition of certain data in a table. With the use of DISTINCT clause data redundancy may be avoided. This clause will eliminate the repetitive appearance of same data. DISTINCT can come only once in a given select statement.

Syntax:

```
SELECT DISTINCT <column_name>
FROM <table_name>
  WHERE <conditions>;
```

Example

Problem Statement:- Retrieve Different Distinct release movie country .

```
SELECT Distinct
    mov_rel_country
FROM
    tbl_movies
```

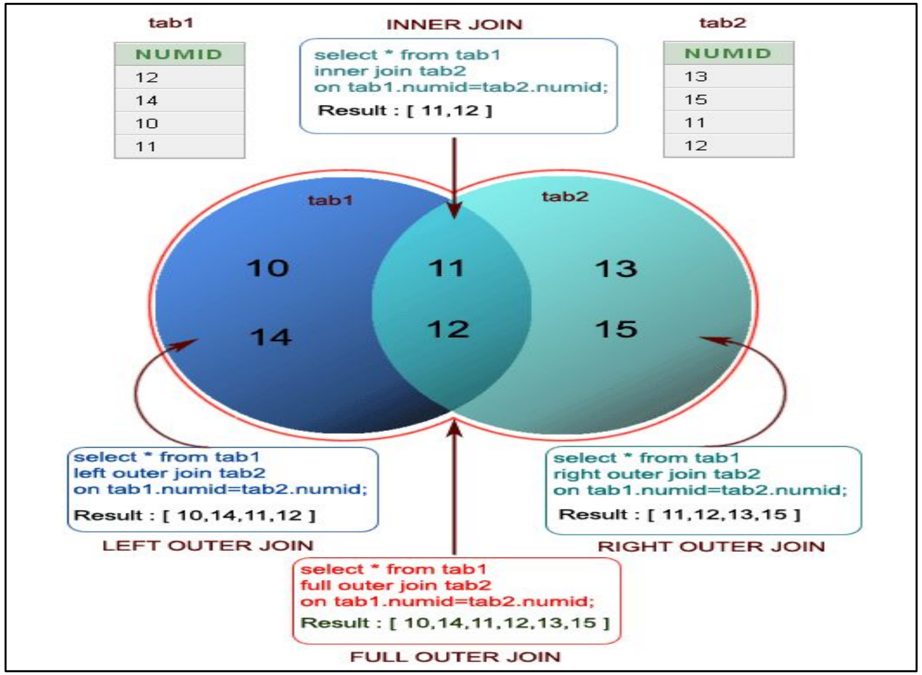
Alias

- An alias is a temporary name given to a table, column, or expression in a query. It is one of the easiest and most common practices used in SQL writing due to its simplicity of use and the meaningful impact it can have on your query.
- A SQL alias is used when you want to make your code cleaner and easier to read and write . An alias *provides a shorthand name for a longer or more complex object name like a table or column*. It is a quick way to simplify query writing by creating custom names for these data objects to improve interpretation.
- An alias can be as simple as an abbreviation of a current name or a brand new one you've created in order to improve the readability of your code.

```
SELECT
    first_name + ' ' + last_name AS full_name
FROM
    sales.customers
ORDER BY
    first_name;
```

Joins

An SQL JOIN clause combines rows from two or more tables. It creates a set of rows in a temporary table.



tab1

NUMID
12
14
10
11

tab2

NUMID
13
15
11
12

INNER JOIN

```
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
Result : [ 11,12 ]
```

LEFT OUTER JOIN

```
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
Result : [ 10,14,11,12 ]
```

RIGHT OUTER JOIN

```
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
Result : [ 11,12,13,15 ]
```

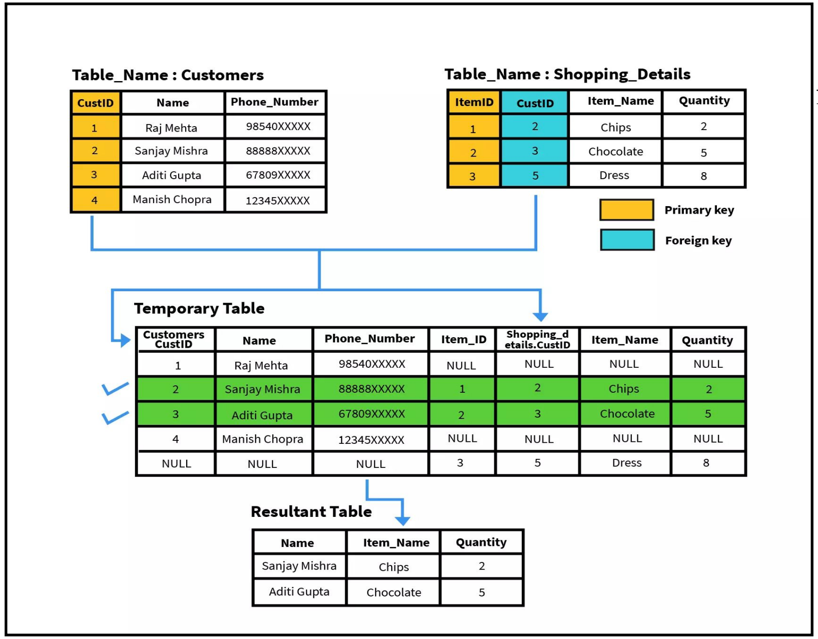
FULL OUTER JOIN

```
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
Result : [ 10,14,11,12,13,15 ]
```

Inner Join

Inner join produces a data set that includes rows from the left table, matching rows from the right table.

Example:-Here are two table



Table_Name : Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

🟨	Primary key
🟦	Foreign key

Temporary Table

	Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
	1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
✓	2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
✓	3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
	4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
	NULL	NULL	NULL	3	5	Dress	8

Resultant Table

Name	Item_Name	Quantity
Sanjay Mishra	Chips	2
Aditi Gupta	Chocolate	5

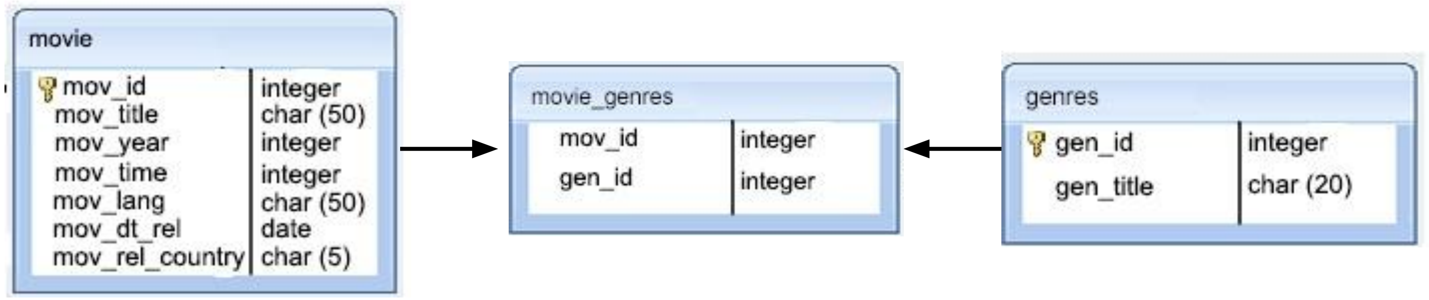
Problem:-Write a query to get all the customers who have bought items from the store. Display their name, item bought, and quantity.

```
SELECT C.Name, S.Item_Name,
S.Quantity
FROM Customers  C INNER JOIN
Shopping_Details  S On
C.CustID==S.CustID;
```

Problem Statement

Write a SQL query to find the movies with year and genres. Return movie title, movie year and generic title.

Here are tables:-

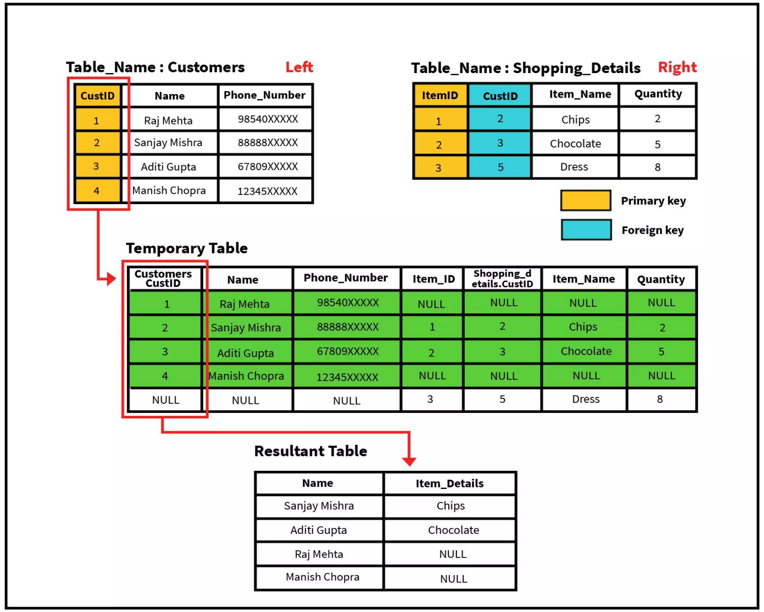


Example

```
SELECT
    m.mov_title AS movie_title,
    m.mov_year AS movie_year,
    g.gen_title AS genre_title
FROM
    tbl_movies m
INNER JOIN
    tbl_movie_geners mg ON m.mov_id = mg.mov_id
INNER JOIN
   tbl_geners g ON mg.gen_id = g.gen_id;
```

Left Join

Left join selects data starting from the left table and matching rows in the right table. The left join returns all rows from the left table and the matching rows from the right table. If a row in the left table does not have a matching row in the right table, the columns of the right table will have nulls.



Table_Name : Customers Left

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details Right

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

🟨	Primary key
🟦	Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8

Resultant Table

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
Raj Mehta	NULL
Manish Chopra	NULL

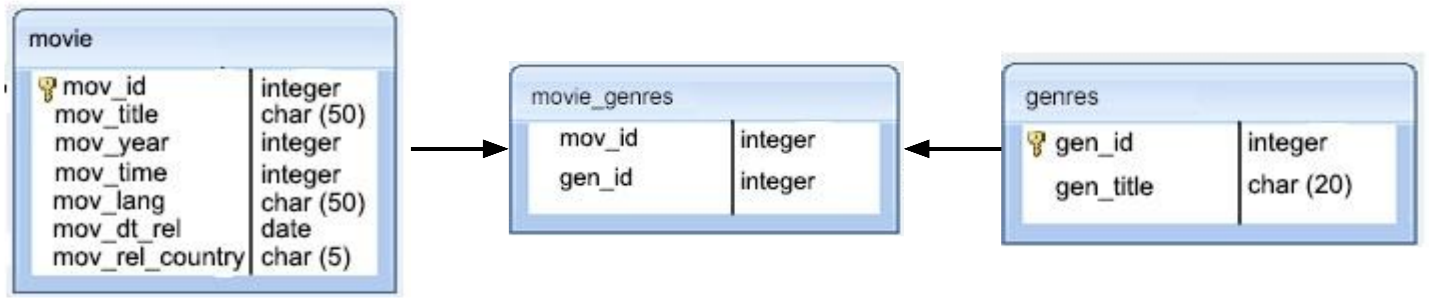
Problem:-Write a query to display all customers irrespective of items bought or not. Display the name of the customer, and the item bought. If nothing is bought, display NULL.

```
SELECT C.Name, S.Item_Name,
S.Quantity
FROM Customers  C LEFTIJOIN
Shopping_Details  S On
C.CustID==S.CustID;
```

Problem Statement

Retrieve a list of movies and their genres, including movies without assigned genres

Here are the Tables

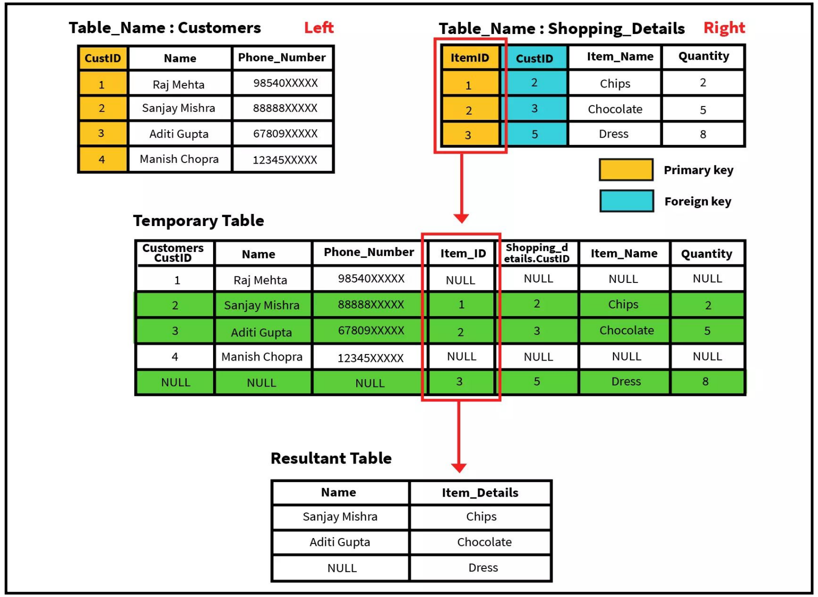


Example

```
SELECT m.mov_id,
m.mov_title,
g.gen_title
FROM tbl_movies m
LEFT JOIN tbl_movie_geners mg ON m.mov_id = mg.mov_id
LEFT JOIN tbl_geners g ON mg.gen_id = g.gen_id;
```

Right Join

The right join or right outer join selects data starting from the right table. It is a reversed version of the left join.
The right join returns a result set that contains all rows from the right table and the matching rows in the left table. If a row in the right table does not have a matching row in the left table, all columns in the left table will contain nulls.



Table_Name : Customers **Left**

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details **Right**

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

🟨 Primary key

🟦 Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8

Resultant Table

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
NULL	Dress

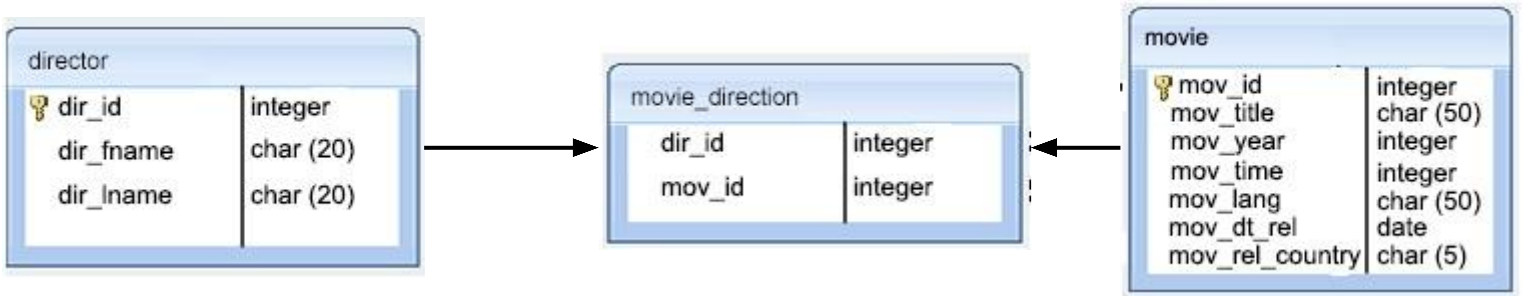
Problem :-Write a query to get all the items bought by customers, even if the customer does not exist in the Customer database. Display customer name and item name. If a customer doesn't exist, display NULL.

```
SELECT C.Name, S.Item_Name,
S.Quantity
FROM Customers  C RIGHT JOIN
Shopping_Details  S On
C.CustID==S.CustID;
```

Problem Statement

Write a SQL query to list All Directors with movie Details and Return dir_Id,dir_fname,dir_lname,movie_id, movie_tile,movie_ year

Here are the Tables:



director

dir_id	integer
dir_fname	char (20)
dir_lname	char (20)

movie_direction

dir_id	integer
mov_id	integer

movie

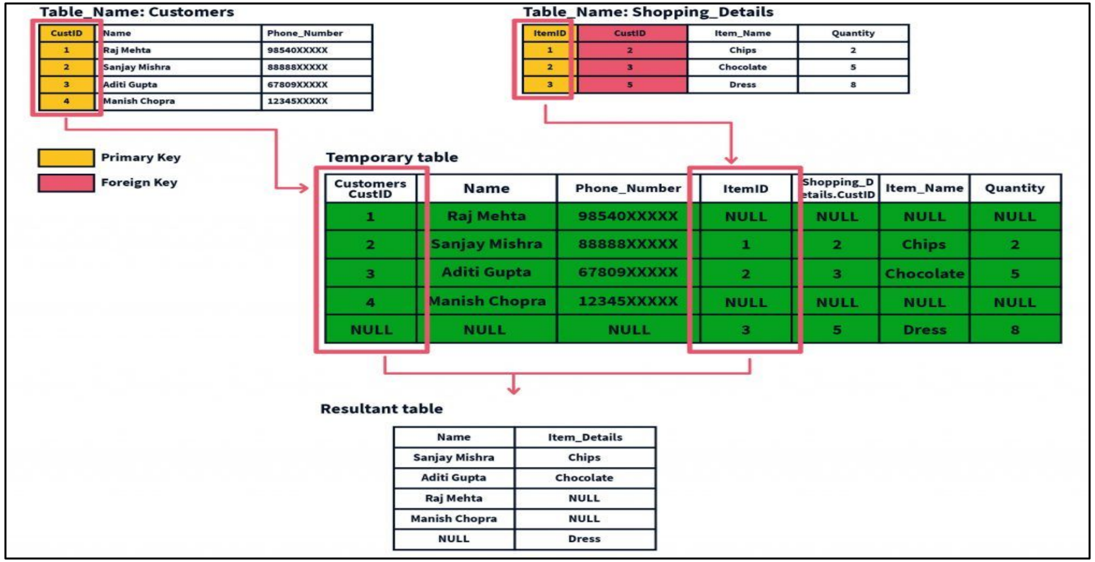
mov_id	integer
mov_title	char (50)
mov_year	integer
mov_time	integer
mov_lang	char (50)
mov_dt_rel	date
mov_rel_country	char (5)

Example

```
SELECT
    d.dir_id,
    d.dir_fname,
    d.dir_lname,
    m.mov_id,
    m.mov_title,
    m.mov_year
FROM
    tbl_directors d
RIGHT JOIN
    tbl_movie_direction md ON d.dir_id = md.dir_id
RIGHT JOIN
    tbl_movies m ON md.mov_id = m.mov_id;
```

Full Outer Join

- The full outer join (a.k.a. SQL Full Join) first adds all the rows matching the stated condition in the query and then adds the remaining unmatched rows from both tables. We need two or more tables for the join.
- After the matched rows are added to the output table, the unmatched rows of the left-hand table are added with subsequent NULL values, and then unmatched rows of the right-hand table are added with subsequent NULL values.



Table_Name: Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name: Shopping_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

🟨	Primary Key
🟥	Foreign Key

Temporary table

Customers CustID	Name	Phone_Number	ItemID	Shopping_Details.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8

Resultant table

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
Raj Mehta	NULL
Manish Chopra	NULL
NULL	Dress

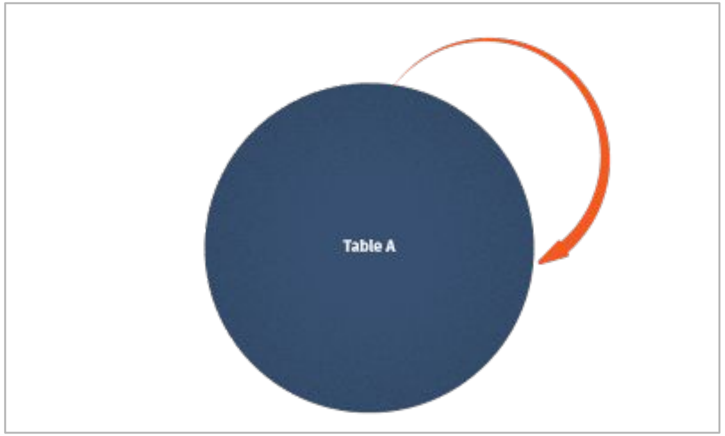
<u>Problem Statement</u>

Write a query to provide data for all customers and items ever bought from the store. Display the name of the customer and the item name. If either data does not exist, display NULL.

```
SELECT c.Name,s.Item_Name
FROM Customers c FULL
OUTER JOIN Shopping_Details
s
ON c.ID = s.ID;
```

SELF JOIN

The SELF JOIN in SQL, as its name implies, is used to join a table to itself. This means that each row in a table is joined to itself and every other row in that table. However, referencing the same table more than once within a single query will result in an error. To avoid this, SQL SELF JOIN aliases are used.



Syntax:-

```
SELECT
   select_list
FROM
   T t1
[INNER | LEFT]  JOIN T t2 ON
   join_predicate;
```

Example

Problem Statement:**-Find Movies Directed by the Same Director**

```
SELECT m1.mov_title AS Movie1, m2.mov_title AS Movie2, d.dir_fname, d.dir_lname
FROM
    tbl_movie_direction md1
INNER JOIN
    tbl_movie_direction  md2 ON md1.dir_id = md2.dir_id AND md1.mov_id <>
md2.mov_id
INNER JOIN
    tbl_movies m1 ON md1.mov_id = m1.mov_id
INNER JOIN
    tbl_movies m2 ON md2.mov_id = m2.mov_id
INNER JOIN
    tbl_directors d ON md1.dir_id = d.dir_id
ORDER BY d.dir_lname, d.dir_fname, m1.mov_title, m2.mov_title;
```

Cross Join

The Cartesian Join, a.k.a. Cross Join, is the cartesian product of all the rows of the first table with all the rows of the second table. Let's say we have m rows in the first table and n rows in the second table. Then the resulting cartesian join table will have m*n rows. This usually happens when the matching column or WHERE condition is not specified.

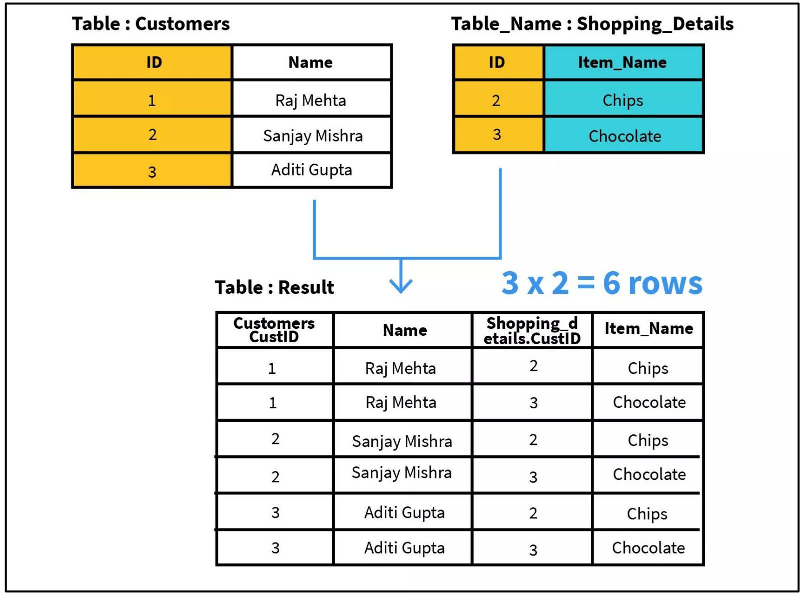


Table : Customers

ID	Name
1	Raj Mehta
2	Sanjay Mishra
3	Aditi Gupta

Table_Name : Shopping_Details

ID	Item_Name
2	Chips
3	Chocolate

Table : Result

Customers CustID	Name	Shopping_details.CustID	Item_Name
1	Raj Mehta	2	Chips
1	Raj Mehta	3	Chocolate
2	Sanjay Mishra	2	Chips
2	Sanjay Mishra	3	Chocolate
3	Aditi Gupta	2	Chips
3	Aditi Gupta	3	Chocolate

Problem Statement

Write a query to give the cartesian product of the Customers and Shopping_Details tables.

```
SELECT *
FROM Customers CROSS JOIN
Shopping_Details;
```

Example

Problem Statement:- Generate All Possible Director-Genre Combinations

```
-- Generate all possible director-genre combinations
SELECT
    d.dir_fname AS DirectorFirstName,
    d.dir_lname AS DirectorLastName,
    g.gen_title AS GenreTitle
FROM
    tbl_directors d
CROSS JOIN
    tbl_geners g
ORDER BY
    d.dir_lname, d.dir_fname, g.gen_title;
```

Example Of Joins-1

Write a SQL query to find out which actors have not appeared in any movies between 1990 and 2000 (Begin and end values are included.). Return actor first name, last name, movie title and release year.

```
SELECT act_fname, act_lname, mov_title, mov_year
FROM tbl_actors
INNER JOIN movie_cast
ON actor.act_id=movie_cast.act_id
INNER JOIN tbl_movies
ON movie_cast.mov_id=movie.mov_id
WHERE mov_year NOT BETWEEN 1990 and 2000;
```

Example Of Joins-2

Write a SQL query to calculate the average movie length and count the number of movies in each genre. Return genre title, average time and number of movies for each genre.

```
SELECT g.gen_title, AVG(m.mov_time) AS avg_mov_time,
COUNT(g.gen_title) AS gen_count
FROM tbl_movies m
INNER JOIN tbl_movie_genres mg ON m.mov_id = mg.mov_id
INNER JOIN tbl_genres g ON mg.gen_id = g.gen_id
GROUP BY g.gen_title;
```

Example Of Joins-3

Write a Query Find directors who have directed at least two movies

```
SELECT d.dir_id, d.dir_fname, d.dir_lname, COUNT(m.mov_id) AS MovieCount
FROM
    tbl_directors d
INNER JOIN
    tbl_movie_direction md ON d.dir_id = md.dir_id
INNER JOIN
    tbl_movies m ON md.mov_id = m.mov_id
GROUP BY
    d.dir_id, d.dir_fname, d.dir_lname
HAVING
    COUNT(m.mov_id) >= 2;
```

Subquery

- subquery is a **query nested within another query** such as **SELECT, INSERT, UPDATE or DELETE**. Also, a subquery **can be** nested within another subquery.
- subquery is called an **inner query** while the query that contains the subquery is called an **outer query**. A subquery can be used anywhere that expression is used and **must be closed in parentheses**.
- subquery is most commonly used with **WHERE and FROM** clause, **IN and NOT IN** operators, **EXISTS and NOT EXISTS.**

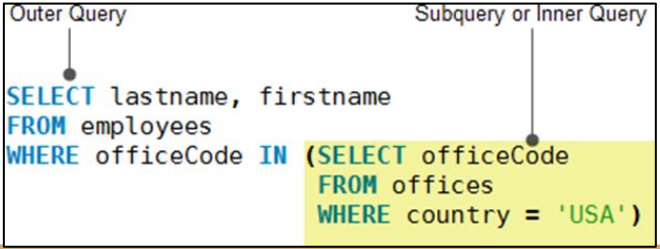
Subquery Contd.

Return employees who work in the offices located in the USA Using Sub-Query would look something like in the following example.

```
SELECT lastName, firstName FROM employees
    WHERE officeCode IN (SELECT officeCode FROM
    offices WHERE country = 'USA’);
```

The subquery returns all office codes of the offices located in the USA.

The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.



```
SELECT lastname, firstname
FROM employees
WHERE officeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

SubQuery With Insert

Problem Statement:-Insert a new movie record into the movie table while also associating it with an existing genre. Let’s Suppose Insert New Movie Records

```
INSERT INTO tbl_movies(mov_title, mov_year, mov_time, mov_lang, mov_dt_rel,
mov_rel_country)
VALUES ('New Action Movie',2023,120,'English','2023-01-01','US');
```

Use a nested query to fetch the gen_id for a specific genre ('Action')

```
INSERT INTO movie_genres (mov_id, gen_id)
VALUES (
    (SELECT TOP 1 mov_id FROM movie WHERE mov_title = 'New Movie'),
    (SELECT gen_id FROM genres WHERE gen_title = 'Action')
);
```

SubQuery Contd.

Problem Statement:-write a SQL query to determine those years in which there was at least one movie that received a rating of at least three stars. Sort the result-set in ascending order by movie year. Return movie year.

Solution:-

```
-- Selecting distinct movie years from the 'movie' table
-- Using a subquery to find mov_id from the 'rating' table based on a condition
-- Specifically, finding mov_id where rev_stars (rating stars) are greater than 3
-- Ordering the result by mov_year in ascending order
SELECT DISTINCT mov_year
FROM tbl_movies
WHERE mov_id IN (
  -- Subquery to find mov_id based on the condition of rev_stars
  SELECT mov_id
  FROM tbl_rating
  WHERE rev_stars > 3
)
ORDER BY mov_year;
```

SubQuery Contd

Problem Statement:- Write a Query to Find genres title that have multiple movies.

Solution:-

```
SELECT  gen_id AS GenreID, gen_title AS GenreName
FROM tbl_geners
    WHERE gen_id IN (
                 SELECT gen_id  FROM  tbl_movie_geners
                 GROUP BY
            gen_id
        HAVING
            COUNT(DISTINCT mov_id) > 1);
```

Joins VS. SubQuery

- JOINs are faster than a subquery and it is very rare that the opposite.
- In JOINs the RDBMS calculates an execution plan, that can predict, what data should be loaded and how much it will take to processed and as a result this process save some times, unlike the subquery there is no pre-process calculation and run all the queries and load all their data to do the processing.
- A JOIN is checked conditions first and then put it into table and displays; where as a subquery take separate temp table internally and checking condition.
- When joins are using, there should be connection between two or more than two tables and each table has a relation with other while subquery means query inside another query, has no need to relation, it works on columns and conditions.

Example

Write a SQL query to determine those years in which there was at least one movie that received a rating of at least three stars. Sort the result-set in ascending order by movie year. Return movie year.

```
SELECT DISTINCT mov_year
FROM movie
WHERE mov_id IN (
  -- Subquery to find mov_id
based on the condition of
rev_stars
  SELECT mov_id
  FROM rating
  WHERE rev_stars > 3
)
ORDER BY mov_year;
```

```
SELECT DISTINCT m.mov_year
FROM movie m
INNER JOIN rating r ON m.mov_id =
r.mov_id
WHERE r.rev_stars > 3
ORDER BY m.mov_year;
```

Union vs Union All

UNION operator allows you to combine two or more result sets of queries into a single result set.

To combine result set of two or more queries using the UNION operator, these are the basic rules that you must follow for each SELECT statement must have

•*The same number of Columns selected and same number of columns expressions,*
•*The same datatype and*
•*Have them in the same order,*
•***But they need not have to be in in the same length.***

Note: By default, the UNION operator removes duplicate rows even if you don’t specify the DISTINCT operator explicitly.

Union vs Union All Contd.

ExampleTables

- *DROP TABLE IF EXISTS t1;*
- *DROP TABLE IF EXISTS t2;*
- *CREATE TABLE t1 (id INT PRIMARY KEY);*
- *CREATE TABLE t2 (id INT PRIMARY KEY);*
- *INSERT INTO t1 VALUES (1),(2),(3);*
- *INSERT INTO t2 VALUES (2),(3),(4);*

Union

```
SELECT id FROM
t1 UNION SELECT
id FROM t2;
```

Union All

```
SELECT id FROM t1
UNION ALL SELECT id
FROM t2;
```

Example

Problem Statement:-Write a SQL query to find the name of all reviewers and movies together in a single list.

```
-- Selecting the 'rev_name' column from the 'reviewer' table
-- Combining the result with the 'mov_title' column from the 'movie' table using UNION
SELECT tbl_reviewer.rev_name
FROM tbl_reviewer
UNION
SELECT tbl_movies.mov_title
FROM tbl_movies;
```

Temp Table

- Temporary table is a special type of table that allows you to store a temporary result set, which you can reuse several times in a single session.
- A temporary table is very handy when it is impossible or expensive to query data that requires a single SELECT statement with the JOIN clauses. In this case, you can use a temporary table to store the immediate result and use another query to process it.
- A temporary table has the following specialized features:
  - *A temporary table is created by using **CREATE TEMPORARY TABLE** statement.*
  - *MySQL removes the temporary table automatically **when the session ends** or **the connection is terminated**. Of course, you can use the DROP TABLE statement to remove a temporary table explicitly when you are no longer use it.*
  - *A temporary table is only **available** and **accessible** to the **client that creates it**. **Different clients can create temporary tables with the same name without causing errors** because only the client that creates the temporary table can see it. **However, in the same session, two temporary tables cannot share the same name**.*
  - *A temporary table can have the same name as a normal table in a database.*
- For example, if you create a temporary table named employees in the sample database, the existing employees table becomes inaccessible.
- Every query you issue against the employees table is now referring to the temporary table employees. When you drop the employees temporary table, the permanent employees table is available and accessible.

Temp Table Contd.

Creating a temporary table whose structure based on a query example

```
CREATE TABLE #TempMovieGenres (mov_id INT,mov_title
VARCHAR(255),mov_year INT,mov_time INT,mov_lang
VARCHAR(50),mov_dt_rel DATE,mov_rel_country VARCHAR(100),gen_id
INT,gen_title VARCHAR(255));
INSERT INTO #TempMovieGenres
SELECT
m.mov_id,m.mov_title,m.mov_year,m.mov_time,m.mov_lang,m.mov_dt_rel,m.
mov_rel_country,mg.gen_id,g.gen_title
FROM
    movie m
JOIN
    movie_genres mg ON m.mov_id = mg.mov_id
JOIN
    genres g ON mg.gen_id = g.gen_id;
```

TempMovieGenres temporary table like querying from a permanent table

```
Select * from #TempMovieGenres
```

CTE Recursive

- A common table expression is a named temporary result set that exists only within the execution scope of a single SQL statement e.g., SELECT, INSERT, UPDATE, or DELETE. Similar to a derived table, a CTE is not stored as an object and last only during the execution of a query.
- Unlike a derived table, a CTE can be self-referencing (a recursive CTE) or can be referenced multiple times in the same query. In addition, a CTE provides better readability and performance in comparison with a derived table.
- The structure of a CTE includes the name, an optional column list, and a query that defines the CTE. After the CTE is defined, you can use it as a view in a SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.
  – `WITH expression_name[(column_name [,...])] AS (CTE_definition)SQL_statement;`
- Notice that the number of columns in the query must be the same as the number of columns in the column_list. If you omit the column_list, the CTE will use the column list of the query that defines the CTE

CTE Recursive

There are some contexts that you can use the WITH clause to make common table expressions:

First, specify the expression name (`expression_name`) to which you can refer later in a query.

```
WITH expression_name[(column_name [,...])]
```

Next, specify a list of comma-separated columns after the expression_name. The number of columns must be the same as the number of columns defined in the `CTE_definition`

```
    (CTE_definition)
```

Finally, refer to the common table expression in a query (`SQL_statement`) such as `SELECT`, `INSERT`, `UPDATE`, `DELETE`, or `MERGE`.

```
SQL_statement;
```

CTE Recursive Contd.

Example:

```
WITH MovieDirectors AS (
    SELECT m.mov_id,m.mov_title,d.dir_firstname,d.dir_lastname
          FROM tbl_movies  m
    INNER JOIN movie_direction md ON m.mov_id = md.mov_id
    INNER JOIN director d ON md.dir_id = d.dir_id);

Select * from MovieDirectors Where WHERE
    dir_firstname = 'Christopher'
    AND dir_lastname = 'Nolan';
```

Working

The CTE named MovieDirectors is defined in the WITH clause.It selects columns from the movie, movie_direction, and director tables to get information about movies and their directors.