Top-n Analysis

- Top-n queries ask for the n largest or smallest values of a column, for example
  - Top ten students of a class
  - The ten worst selling product or ten best selling products
- Both largest and smallest values sets are considered top-n queries

SELECT [column_list], ROWNUM FROM **(Select [column_list] From table ORDER BY Top-n_column)** WHERE ROWNUM <= N; Where clause specifies the n rows to be returned.	A subquery or an inline view to generated the sorted list of data. For results retrieving the largest values , a DESC parameter is needed. The outer query contains ROWNUM, which assigns a sequential values starting with 1 to each returned row.

Some important points to remember:

Sort by Order By clause :

- Col. alias can be used in order by clause
- Sort by col. that is not in the SELECT list is possible
- Default sorting order ASC
- Possible order clause ASC/ DESC

- Character and date values are enclosed with single quotation
- Character values are case sensitive and date values are format sensitive
- The default date format is **DD-MON-RR**

SQL functions can be used in the SELECT :

```
SELECT empID, CONCAT(firstName, lastName) Name, Length(lastName)
       INSTR(lastName, 'a') "Contains 'a' ?" , lastName FROM empMaster
       WHERE SUBSTR(lastName, -1, 1) = 'n';
```

Last name contains a and end with n

empID	Name	Length(lastName)	Contains 'a'?	lastName
102	KexDe Haan	7	5	De Haan
200	JenniferWhalen	6	3	Whalen
201	MichaelHartstein	9	2	Hartstein

The Dual Table

- The Dual table is owned by the user SYS and can be accessed by all users. It is a 1 col 1 row table with column name DUMMY and contains the value X.
- The Dual table is useful when you want to return a value once only : for instance, a value of a constant, pseudo column, or expression that is not derived from a table with user data.
- The Dual table is generally used for SELECT clause syntax completeness, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from actual tables.

Retrieving data from more than one table (INNER and OUTER join)

- To avoid a Cartesian product, always include a valid join condition in a where clause
- If the Row does not satisfy the join condition it will not appear in the result (for inner/plain join)

Types of Join

Oracle Proprietary Joins (8i and prior)	**SQL : 1999 compliant Joins**
• Equijoin • Nonequijoin • Outer join • Self join 8i and prior join syntax was different from the ANSI standards.	• Cross joins • Natural joins • Using clause • Full or two sided outer joins • Arbitrary join conditions for outer joins 9i and above follow ANSI standard

Oracle Syntax

SELECT table1.col, table2.col FROM table1, table2 WHERE table1.col1 = table2.col2	• **Join condition in the Where clause** • Prefix the col. Name with the table name when the same col. name appears in more than one table • To join n tables together, you need a minimum of n-1 join condition
Equijoin with additional search condition SELECT emp.empID, emp.lastName, emp.deptID, dept.deptID, dept.deptName FROM emp, dept WHERE emp.deptID = dept.deptID AND lastName = 'Kumar'	• WHERE specify the how the table are to be joined • As deptID is common in both tables , it needs to be prefixed. • Additional search condition in WHERE clause

Qualifying Ambiguous column name

- Use table prefixes to qualify column names that are in multiple tables
- Improve performance by using table prefixes - if no common col. Names, no need to use table prefix. However using table prefix improves performance, because you tell the Oracle Server exactly where to find the columns
- Distinguish columns that have identical names but reside in different tables by col. Aliases.
- Simplify queries by using **table aliases** - Qualifying col name by table names can be very time consuming (if table name is lengthy). Table aliases help to keep SQL code smaller, therefore using less memory.
    - Alias maxi. Length 30 ch, but shorter is the better

- If table alias is used, use that alias instead of the table name throughout the SELECT
- This is the **most preferred option**

```
SELECT a.empID, a.lastName,  a.deptID,  b.deptID,  b.deptName
FROM  emp  a ,  dept  b
WHERE   a.deptID = b.deptID AND  lastName = 'Kumar'
```

Joining More than two tables

Additional Search Condition

```
SELECT a.empID, a.lastName,  a.deptID,  b.deptID,  b.deptName,  c.city
FROM  emp  a ,  dept  b ,  locations  c
WHERE   a.deptID = b.deptID  AND  b.locationID = c.locationID AND  a.lastName = 'Kumar'
```

Non-Equijoin

```
SELECT  e.lastName,  e.salary,  j.gradeLevel
FROM   emp  e,  jobGrade  j
WHERE  e.salary  BETWEEN  j.lowestSal  AND  j.highestSal  ;
```

Self Join

```
Select  a.lastNAme  ||  'works for '  ||  b.lastName
FROM  employee  a,  employee b
WHERE  a.mgrID = b.empID
```

SQL :1999 Syntax

```
SELECT   table1.column, table2.column
FROM  table1
[CROSS JOIN  table2]  |
[NATURAL JOIN table2 ] |
[JOIN table2  USING (column)] |
[JOIN table2   ON (table1.column1 = table2.column2) ]  |
[LEFT | RIGHT | FULL   OUTER JOIN  table2  ON ( table1.column1 = table2.column2) ]
```

table1.column	Denotes the table and column from which data is retrieved
CROSS JOIN	Return Cartesian product
NATURAL JOIN	Join two table based on the same column name
JOIN table USING …..	Perform an equijoin based on the column name
JOIN table on …..	Perform an equijoin based on the condition in the ON clause
LEFT/RIGHT/FULL OUTER	• A join of two tables returning only matching rows is an inner join. • Outer join is to see rows that **do not meet the join condition** as well as the rows **meet the join condition.**

	• **Left** Outer Join - A join between two tables that returns the results of an inner join as well as unmatched rows from left. • **Right** Outer Join - A join between two tables that returns the results of an inner join as well as unmatched rows from right. • A join between two tables that returns the results of the inner join as well as the results of a left and right join is a **full** outer join.

Creating Joins with the Using Clause

- Natural joins use all columns with **matching names** and **data types** to join the tables. The USING clause can be used to specify only those columns that should be used for an equijoin.

  Note : Use the USING clause to match only one column when more than one column matches.

- Do not use a table name or alias in the referenced columns (in the USING clause). The same restriction applies to NATURAL join also. Therefore columns that have the same name in both tables have to be used without any qualifier.

- The NATURAL JOIN and USING clause are mutually exclusive.

```
SELECT  l.city , d.deptName
FROM   location l
JOIN   dept   d   USING (locationID)
WHERE loactionID = 1400;
```

Equivalent Equijoin Syntax

```
SELECT  l.city , d.deptNAme
FROM   location l,  dept   d
WHERE   d.locationId   =   l.loactionID   AND
d.locationID = 1400;
```

```
SELECT  l.city , d.deptNAme
FROM   location l
JOIN   dept   d   USING (locationID)
WHERE d.loactionID = 1400;
```

This is invalid because alias is used in where clause

Creating Joins with the ON Clause

- To specify arbitrary conditions or specify columns to join, the ON clause is used
- Helps to separate the join condition from other search conditions. **More clear syntax** results better understanding and maintenance of code.

```
SELECT  l.city , d.deptNAme
FROM   location l
JOIN   dept   d   ON d.locationID = l.locationID
WHERE d.loactionID = 1400;
```

Example of three way join (three table)

```
SELECT  e.empID,  e.empName, l.city ,
        d.deptName
FROM   empMaster e
```

	JOIN dept d ON e.deptID = d.deptID JOIN location l ON d.locationID = l.locationID WHERE d.loactionID = 1400;
In SQL:1999 compliant syntax, joins are performed from **left to right**, so the first join to be performed is **empMaster JOIN dept.** The first join condition can reference columns in **empMaster** and **dept** but cannot reference columns in **location.** The second join condition can reference columns for all three tables. **Above three way join can also be written using Oracle syntax :** SELECT e.empID, e.empName, l.city , d.deptName FROM empMaster e, dept d , location l WHERE e.deptID = d.deptID AND d.locationID = l.locationID AND d.loactionID = 1400;	

Left , Right, Full OUTER JOIN

Old Oracle syntax is not preferred now.

- The outer join operator is the plus sign **(+)** - Oracle Syntax
- Outer join symbol (+) can be placed on either side of the join condition, **but not on both side.**
- Place the outer join symbol following the name of the column in the table **without the matching rows.** The side that has information missing

SELECT e.empID, e.empName, e.deptid, d.deptName FROM empMaster e **LEFT** OUTER JOIN dept d ON e.deptID = d.deptID	All the records from the left table **empMaster** will appear in the result set (matched or unmatched deptID in dept table)
SELECT e.empID, e.empName, e.deptid, d.deptName FROM empMaster e, dept d WHERE d.deptID (+) = e.deptID	**Earlier Oracle Syntax fro Left outer join**
SELECT e.empID, e.empName, e.deptid, d.deptName FROM empMaster e **RIGHT** OUTER JOIN dept d ON e.deptID = d.deptID	Retreives all rows in **dept** table (right table) even if there is no match in the empMaster table.
SELECT e.empID, e.empName, e.deptid, d.deptName FROM empMaster e **FULL** OUTER JOIN dept d ON e.deptID = d.deptID	**Here some employee record will no dept as well as some dept. record will no employee appear in the result set.**
Above query retrieves all records in the empMaster table, even if there is no match in the dept table. It also retrieves all rows in the dept table, even if there is no match in the empMaster table.	

Select the order items and challan information if available :

```
SELECT  f.OrderDate, a.OrderNo ,  a.OrderLineNo, a.ItemId, a.OrderBP,
          a.OrderQty, NVL(c.Quantity,0) ChalanQty, a.ReqDelDate, c.ChalanDate,
          f.CustomerName, CASE  WHEN  a.Priority = 1 THEN  'CST'    WHEN
          a.Priority = 2 THEN  'KEY'   WHEN a.Priority = 3 THEN  'Key + CST'
          ELSE  '' END   Priority
FROM  tblOrderLineItem a
INNER JOIN  tblOrderHeader f    ON a.OrderNo = f.OrderNo AND  f.Hold = 0 AND f.Status <> 'C'
LEFT OUTER JOIN tblChalanLineItem c   ON a.OrderNo = c.OrderNo AND
                                          a.OrderLineNo = c.OrderLineNo
Where  f.OrderRegion = 'West'
        AND a.DeliveryBP IN ( Select BPCode From tblBPMaster Where BPType = 'C' ))
```

Summary using Grop function

Group Functions : Operates on set of rows to give one result per group

Type of Group Functions

ALL means consider all values. It is default if nothing specified consider all values.

Function	
AVG ({ * | [DISTINCT | ALL] expr })	Average value of n, ignoring null values
COUNT ({ * | [DISTINCT | ALL] expr })	Number of rows, where **expr** evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX ([DISTINCT | ALL] expr)	Maximum values of expr , ignoring null values
MIN ([DISTINCT | ALL] expr)	Minimum values of expr , ignoring null values
STDDEV ([DISTINCT | ALL] x)	Standard deviation of n, ignoring null values
SUM ([DISTINCT | ALL] n)	Sum values of n, ignoring null values
VARIANCE ([DISTINCT | ALL] x)	Variance of n, ignoring null values

Example : SELECT SUM(DISTINCT salary) AS "Total Salary" FROM employees WHERE salary > 50000;

Group Function Syntax

Examples below consider the tables as one large group of information.

SELECT [column,] group_fumction(column), …… FROM table [WHERE condition]	**Guidelines** • DISTINCT makes the function consider only non-duplicate values; ALL for considering duplicates. **ALL is default**. • Data types for an **expr** used in the function may be CHAR, VARCHAR2, NUMBER or DATE

[GROUP BY column] [ORDER BY column]	• All group functions ignore null values. To substitute a value for null values, use the NVL, NV2 or COALESCE functions • The Oracle Server implicitly sorts the result set in ascending order when using GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause.

Using AVG, SUM, MAX SELECT Avg(salary), MAX(Salary), Min(Salary), Max(hireDate), Min(HireDate), SUM(salary) FROM empMaster WHERE ……. SELECT AVG(NVL(comPercent,0)) FROM empMAster	SELECT Min(lastNAme), Max(last_name) FROM empMaster • Min and Max can be used for any permissible data types. • Group function will only consider non-null values NVL function forces to include null values (if comPercent is NULL NVL function will return 0)
Count(*) SELECT COUNT(*) FROM empMaster WHERE deptID = 50	• Returns the number of rows satisfying the WHERE clause, including **duplicated** rows and rows containing **null** values in any of the columns.
Count(expr) SELECT COUNT(comper) FROM empMaster WHERE deptID = 50	• Returns the number of rows satisfying the WHERE clause considering only **non-null** values for the expression.
Count(DISTINCT expr) SELECT COUNT(DISTINCT comper) FROM empMaster WHERE deptID = 50	• Returns the number of rows satisfying the WHERE clause considering only **non-null** and **distinct** values for the expression. • Suppress any duplicate values within the column.

Creating Group of Data

- Divide rows in a table into smaller groups by using the GROUP BY clause
- Columns specified in the group by clause determine the basis of grouping the rows
- If group by clause is included in the SELECT clause, you cannot use any column/expression without using any group functions unless the column/expression appears in the GROUP BY clause.
- Where clause will excludes rows **before the group formation**.
- You must include the columns in the GROUP BY clause, no column alias is allowed.
- The Oracle Server implicitly sorts the result set in ascending order when using GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause

SELECT deptID, AVG(salary) AvgSal FROM empMAster GROUP BY deptID ORDER BY AVG(salary) SELECT AVG(salary) AvgSal FROM empMAster GROUP BY deptID	• All columns in the SELECT list that are not in group functions must be in the GROUP BY clause. • Group function can be used in order by clause, but you cannot use **AvgSal** (col alias). Default sorting order- deptID. • Group by column may not be in the select list. But the result is not very meaningful.
Groups within Groups – multiple columns in group by SELECT deptID DeptCode, JobID, SUM(salary) TotalSal FROM empMaster GROUP BY deptID, jobID	• SUM will be applied to the salary column for all job IDs within each deptID • Default sorting order is deptID , jobID
SELECT deptID, COUNT(lastName) FROM empMaster SELECT deptID, COUNT(lastName) FROM empMaster WHERE COUNT(lastName) > 5	**Illegal Query** • **Any column/expression in the select list that is not using any aggregate function must be in the GROUP By clause** • **WHERE clause is not for restricting group**

Excluding Group Result – **Having clause**

- To restrict the group we can use HAVING clause. It will specify which groups are to be displayed based on aggregate function. After the groups are formed, the groups that match the criteria in the HAVING clause are displayed
- Groups are formed and group functions are calculated before the HAVING clause is applied to the group in the SELECT list. Hence it is **recommended** that use HAVING after the GROUP BY clause in SELECT.

SELECT jobID, SUM(salary) Payroll FROM empMaster WHERE jobID NOT LIKE '%REP%' GROUP BY jobID HAVING SUM(salary) > 13000 ORDER BY SUM(salary);	• It is not recommended to use only the aggregated function in HAVING criteria. **Nested Group functions** SELECT MAX(AVG(salary)) FROM empMaster GROUP BY deptID

Example

```
SELECT  '0' AS Flag, a.ItemId, b.OrderRegion, Count(distinct a.OrderNo)
        OrderNo, Sum(a.OrderQty) OrderQty
FROM    tblOrderLineItem a
INNER JOIN tblOrderHeader b                ON a.OrderNo = b.OrderNo
Where a.LineStatus <> 'C' AND b.OrderRegion = 'West')
Group BY a.ItemId, b.OrderRegion
Having  Sum(a.OrderQty) > 50
ORDER  BY  b.OrderRegion  DESC
```

If we don't specify the order by clause the default sorting order will be a.ItemId, a.OrderRegion as we are grouping based on those columns.

UNION INTERSECT AND MINUS

- Relational set operators are used to combine or subtract the records from two tables. These are used in the SELECT query to combine the records or remove the records.
- In order to set operators to work in database, it should have same number of columns participating in the query and the datatypes of respective columns should be same. This is called **Union Compatibility**.
- The resulting records will also have same number of columns and same datatypes for the respective column.

UNION

It combines the similar columns from two or more tables or queries into one resultant table. All columns that are participating in the UNION operation should be **Union Compatible**. This operator combines the records from all the tables into one.

The syntax for the UNION statement is: <query> UNION [ALL] <query>

- In case of UNION , **duplicate records** from result will be **eliminated**. The resulting records will be from both table and **distinct**.
- Two table are said to be union compatible if both the table have same number of attributes (column) and corresponding attributes have the same data type (int,char,float,date etc.). Corresponding attributes means first attributes of both relations, then second and so on.

union compatible:
Table A: (First_name (char), Last_name(char), Date_of_Birth(date))
Table B: (FName(char),LName(char),DOB(date))

Both table have 3 attributes and of same date type.

Not compatible:
A: (First_name (char), Last_name(char), Date_of_Birth(date))
B: (FName(char),LName(char),PhoneNumber(number)) (The third attributes are different.)

- The rows of the participating tables/queries have the same structure, so they can be put in the same final result table.
- What is interesting is that the result of a UNION has no name, and its columns are not named. If you want to have names, then you have to use an AS operator to create those names (alias) , thus.

  ((SELECT a, b, c FROM TableA WHERE city = 'Boston')
  UNION
  (SELECT x, y, z FROM TableB WHERE city = 'New York'))
  AS Cities (tom, dick, harry)

  However, in RDBMS products will find a multitude of other ways of doing this:

  - The columns have the names of the first table in the UNION statement.
  - The columns have the names of the last table in the UNION statement.
  - The columns have the names generated by the SQL engine.
  - The columns are referenced by a position number. This was the SQL-89 convention.

Suppose we have to see the employees in EMP_TEST and EMP_DESIGN tables, we want to combine both the results from tables in to one set using UNION.

```
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_TEST
UNION
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_DESIGN;
```

We can notice that Result will have same column names as first query. Duplicate record – 104 from EMP_TEST and EMP_DESIGN are showed only once in the result set. Records are sorted in the result.

EMP_TEST

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
104	Kathy	Holland	324343

Union

EMP_DESIGN

EMP_ID	ENAME	EMP_ADDRESS	SSN
103	Rose	Freser Town	6744545
102	Marry	Novi	343613
105	Laurry	Rochester Hills	97676
104	Kathy	Holland	324343

UNION

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
102	Marry	Novi	343613
103	Rose	Freser Town	6744545
104	Kathy	Holland	324343
105	Laurry	Rochester Hills	97676

Example : Here is a complex union

```
select a.LABEL,  a.ID,  'Operating System'  as rowtype, a.OS_NAME   as rowvalue1,
        b.SWARE_DESC   as rowvalue2
from COMPUTER  a
inner  join MATCHED_SWARE  b   on a.ID = b.ID
inner  join SWARE_SIG c              on b.SWARE_SIG_ID = c.SWARE_SIG_ID

union all

select a.LABEL , a.ID,  'Processor '   as rowtype, b.PROCESSOR_ID as rowvalue1,
        cast(count(*) as varchar) as rowvalue2
from COMPUTER  a
inner  join PROCESSOR b     on a.ID = b.ID
group  by a.LABEL, a.ID, b.PROCESSOR_ID
```

In this example, the column in the result set called "rowvalue2" will have string values, hence it is necessary to CAST the numeric count as VARCHAR.

UNION ALL

This operation is also similar to UNION, but it **does not eliminate the duplicate records**. It shows all the records from both the tables. All other features are same as UNION. Look at the same example below with UNION ALL operation.

```
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_TEST
UNION ALL
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_DESIGN;
```

Two records with 104 EMP_ID

EMP_TEST

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
104	Kathy	Holland	324343

Union ALL

EMP_DESIGN

EMP_ID	ENAME	EMP_ADDRESS	SSN
103	Rose	Freser Town	6744545
102	Marry	Novi	343613
105	Laurry	Rochester Hills	97676
104	Kathy	Holland	324343

UNION

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
102	Marry	Novi	343613
103	Rose	Freser Town	6744545
104	Kathy	Holland	324343
104	Kathy	Holland	324343
105	Laurry	Rochester Hills	97676

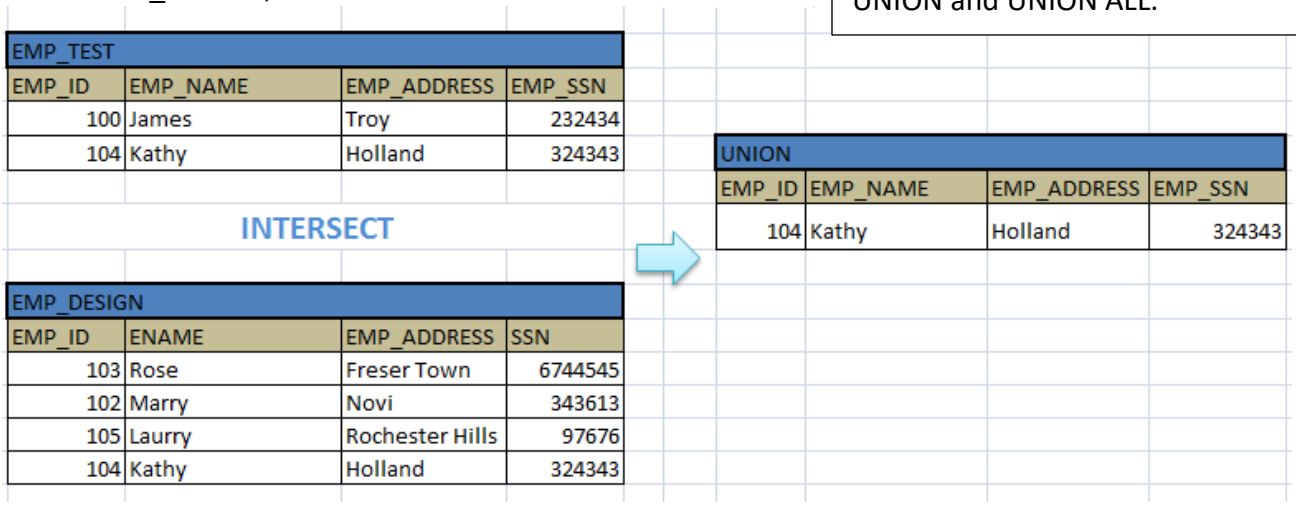
INTERSECT

This operator is used to pick the records from both the tables which are common to them. In other words it picks only the duplicate records from the tables. Even though it selects duplicate records from the table, each duplicate record will be displayed only once in the result set. It should have **UNION Compatible** columns to run the query with this operator.

Same example above when used with INTERSECT operator, gives below result.

```
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_TEST
INTERSECT
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_DESIGN;
```

We have INTERSECT ALL operator too. But it is same as INTERSET. There is no difference between them like we have between UNION and UNION ALL.



EMP_TEST

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
104	Kathy	Holland	324343

INTERSECT

EMP_DESIGN

EMP_ID	ENAME	EMP_ADDRESS	SSN
103	Rose	Freser Town	6744545
102	Marry	Novi	343613
105	Laurry	Rochester Hills	97676
104	Kathy	Holland	324343

UNION

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
104	Kathy	Holland	324343

MINUS

This operator is used to display the records that are present only in the first table or query, and doesn't present in second table / query. It basically subtracts the first query results from the second. Let us see the same example with MINUS operator.

```
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_TEST
MINUS
SELECT EMP_ID, EMP_NAME, EMP_ADDRESS, EMP_SSN
FROM EMP_DESIGN;
```

We can notice only the records that do not exists in EMP_DESIGN are displayed in the result. The record which appears in both the tables is eliminated. Similarly, the records that appear in second query but not in the first query are also eliminated.

EMP_TEST			
EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434
104	Kathy	Holland	324343

MINUS

EMP_DESIGN			
EMP_ID	ENAME	EMP_ADDRESS	SSN
103	Rose	Freser Town	6744545
102	Marry	Novi	343613
105	Laurry	Rochester Hills	97676
104	Kathy	Holland	324343

UNION			
EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_SSN
100	James	Troy	232434