SET OPERATIONS:

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

There are four different types of SET operations :

- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

UNION Operation :

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set.

In case of union, *number of columns and datatype must be same in both the tables*, on which UNION operation is being applied.

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

Let's see the following sample tables: t1 and 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);
```

The following statement combines result sets returned from t1 and t2 tables:

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

The final result set contains the distinct values from separate result sets returned by the queries:

| id |

| 1 |

| 2 |

| 3 |

| 4 |

Because the rows with value 2 and 3 are duplicates, the UNION removed them and kept only unique values.

UNION ALL Operation :

If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. Because UNION ALL does not need to handle duplicates, it performs faster than UNION DISTINCT .

SELECT id

FROM t1

UNION ALL

SELECT id

FROM t2;

Output

| id |

| 1 |

| 2 |

| 3 |

| 2 |

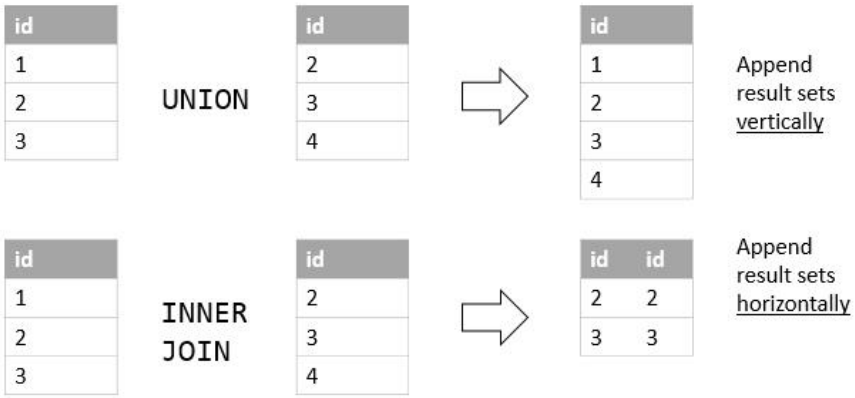
| 3 |

| 4 |

As you can see, the duplicates appear in the combined result set because of the UNION ALL operation.

UNION vs. JOIN

A JOIN combines result sets horizontally, a UNION appends result set vertically. The following picture illustrates the difference between UNION and JOIN:



MINUS :

The MINUS compares the results of two queries and returns distinct rows from the result set of the first query that does not appear in the result set of the second query.

The following illustrates the syntax of the MINUS operator:

SELECT select_list1

FROM table_name1

MINUS

SELECT select_list2

FROM table_name2;

The basic rules for a query that uses MINUS operator are the following:

- The number and order of columns in both select_list1 and select_list2 must be the same.
- The data types of the corresponding columns in both queries must be compatible.

Suppose that we have two tables t1 and t2 with the following structure and data:

CREATE TABLE t1 (

Id INT PRIMARY KEY

);

CREATE TABLE t2 (

Id INT PRIMARY KEY

);

INSERT INTO t1 VALUES (1),(2),(3);

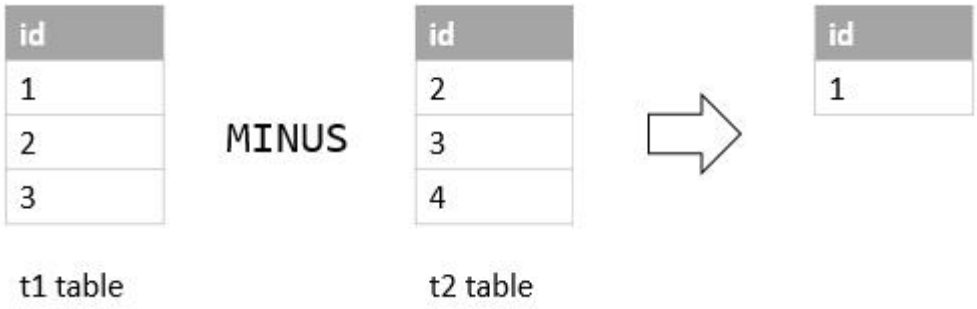
INSERT INTO t2 VALUES (2),(3),(4);

The following query returns distinct values from the query of the t1 table that is not found in the result of the query of the t2 table.

SELECT id FROM t1

MINUS

SELECT id FROM t2;



Note: MySQL does not support MINUS operator

INTERSECT :

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.

Example of Intersect

The First table,

ID	**NAME**
1	abhi
2	adam

The Second table,

ID	NAME
2	adam
3	Cheste

Intersect query will be,

*SELECT * FROM First*

INTERSECT

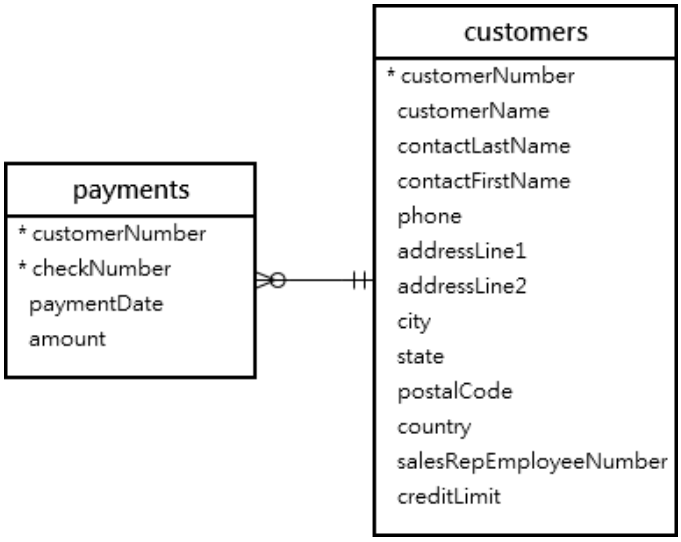
*SELECT * FROM Second;*

The resultset table will look like

ID	NAME
2	adam

Introduction to Views:

Let's see the following tables *customers* and *payments*.




```
SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
```

```
    customers
INNER JOIN
    payments USING (customerNumber);
```

This query returns data from both tables customers and payments using the inner join:

customerName	checkNumber	paymentDate	amount
Atelier graphique	HQ336336	2004-10-19	6066.78
Atelier graphique	JM555205	2003-06-05	14571.44
Atelier graphique	OM314933	2004-12-18	1676.14
Signal Gift Stores	BO864823	2004-12-17	14191.12
Signal Gift Stores	HQ55022	2003-06-06	32641.98
Signal Gift Stores	ND748579	2004-08-20	33347.88
Australian Collectors, Co.	GG31455	2003-05-20	45864.03
Australian Collectors, Co.	MA765515	2004-12-15	82261.22
Australian Collectors, Co.	NP603840	2003-05-31	7565.08
Australian Collectors, Co.	NR27552	2004-03-10	44894.74
La Rochelle Gifts	DB933704	2004-11-14	19501.82
La Rochelle Gifts	LN373447	2004-08-08	47924.19

Next time, if you want to get the same information including customer name, check number, payment date, and amount, you need to issue the same query again.

One way to do this is to save the query in a file, either .txt or .sql file so that later you can open and execute it from MySQL Workbench or any other MySQL client tools.

A better way to do this is to save the query in the database server and assign a name to it. This named query is called a **database view,** or simply, **view**.

A view provides a mechanism to hide certain data from the view of certain users.

Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a view.

In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)

Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

select ID, name, dept_name from instructor

CREATE VIEW

A view is defined using the create view statement which has the form

create view v as < query expression >

where <query expression> is any legal SQL expression.
The view name is represented by v.

Example:

Creating a view stu based on student table.

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

Query to Create a View from the above table will be,

```
CREATE VIEW sale_view
AS
SELECT * FROM Sale WHERE customer = 'Alex';
```

The data fetched from SELECT statement will be stored in another object called **sale_view**.

DISPLAY VIEW

The syntax for displaying the data in a view is similar to fetching data from a table using a SELECT statement.

```
SELECT * FROM sale_view;
```

Output:

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
15	ord5	2000	Alex

Update a VIEW

UPDATE command for view is same as for tables.

Syntax:

```
UPDATE view-name SET VALUE WHERE condition;
```

Example:

```
UPDATE sale_view SET previous_balance=100 WHERE customer=Alex;
```

Drop View

DROP VIEW statement is used to remove one or more views.

Syntax :

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
```

Example:

```
DROP  VIEW sale_view;
```