SQL Joins, Subqueries & Transactions

JOIN Operation

It is used to fetch data from more than one table using a single SQL query. Joins are of multiple types

1. Cross Join
2. Inner Join
3. Left Outer Join
4. Right Outer Join
5. Self Join

For all types of join operations, we are going to use following tables

```
Table-name: category
cat_id  cat_name
1       Electronics
2       Stationary
3       Furniture
4       Food Items
5       House-keeping Goods

Table-name: product
pro_id  pro_name       MRP       MFG_DATE    category_id
1       Parker Pen     349.00    2022-06-16  2
2       Student Chair  1499.00   2021-12-31  3
```

```
3       Dark Chocolate 399.00    2022-04-30  4
4       Microwave Oven 10500.00  2022-05-15  1
5       AC             34500.00  2022-05-01  1
6       Footware       599.00    2022-05-01  NULL
```

Cross Join

This is Father of all join operations.

In the result of cross join, Degree of both tables is added and cardinality of both tables is multiplied

```
SELECT projection FROM left-table CROSS JOIN right-table;
  OR
SELECT projection FROM left-table, right-table;
```

An Example

```
SELECT * FROM Category CROSS JOIN Product;
OR
SELECT * FROM Category, Product;

cat_id  cat_name            pro_id  pro_name       MRP       MFG_DATE    category_id
1       Electronics         1       Parker Pen     349.00    2022-06-16  2
2       Stationary          1       Parker Pen     349.00    2022-06-16  2
3       Furniture           1       Parker Pen     349.00    2022-06-16  2
4       Food Items          1       Parker Pen     349.00    2022-06-16  2
5       House-keeping Goods 1       Parker Pen     349.00    2022-06-16  2
1       Electronics         2       Student Chair  1499.00   2021-12-31  3
2       Stationary          2       Student Chair  1499.00   2021-12-31  3
3       Furniture           2       Student Chair  1499.00   2021-12-31  3
4       Food Items          2       Student Chair  1499.00   2021-12-31  3
5       House-keeping Goods 2       Student Chair  1499.00   2021-12-31  3
1       Electronics         3       Dark Chocolate 399.00    2022-04-30  4
2       Stationary          3       Dark Chocolate 399.00    2022-04-30  4
3       Furniture           3       Dark Chocolate 399.00    2022-04-30  4
4       Food Items          3       Dark Chocolate 399.00    2022-04-30  4
5       House-keeping Goods 3       Dark Chocolate 399.00    2022-04-30  4
1       Electronics         4       Microwave Oven 10500.00  2022-05-15  1
2       Stationary          4       Microwave Oven 10500.00  2022-05-15  1
3       Furniture           4       Microwave Oven 10500.00  2022-05-15  1
4       Food Items          4       Microwave Oven 10500.00  2022-05-15  1
5       House-keeping Goods 4       Microwave Oven 10500.00  2022-05-15  1
1       Electronics         5       AC             34500.00  2022-05-01  1
2       Stationary          5       AC             34500.00  2022-05-01  1
3       Furniture           5       AC             34500.00  2022-05-01  1
4       Food Items          5       AC             34500.00  2022-05-01  1
5       House-keeping Goods 5       AC             34500.00  2022-05-01  1
1       Electronics         6       Footware       599.00    2022-05-01  NULL
2       Stationary          6       Footware       599.00    2022-05-01  NULL
3       Furniture           6       Footware       599.00    2022-05-01  NULL
4       Food Items          6       Footware       599.00    2022-05-01  NULL
5       House-keeping Goods 6       Footware       599.00    2022-05-01  NULL
```

Inner Join

It is used to take common entries from the participating tables

```
SELECT projection FROM left-table INNER JOIN right-table
ON left-table.col-name = right-table.col-name;

  OR

SELECT projection FROM left-table, right-table
WHERE left-table.col-name = right-table.col-name;
```

💡 Tip: Both the queries will produce same result but the former query is having better performance because in the latter query cross join will be performed then WHERE clause will filter the record so all that will take lot of time than the former syntax. Use of former syntax is preferred

An Example

```
SELECT * FROM category INNER JOIN product
ON category.cat_id = product.category_id;

  OR

SELECT * FROM Category, Product
WHERE Category.cat_id = Product.pro_id;

cat_id  cat_name             pro_id  pro_name       MRP      MFG_DATE    category_id
2       Stationary           1       Parker Pen     349.00   2022-06-16  2
3       Furniture            2       Student Chair  1499.00  2021-12-31  3
4       Food Items           3       Dark Chocolate 399.00   2022-04-30  4
1       Electronics          4       Microwave Oven 10500.00 2022-05-15  1
1       Electronics          5       AC             34500.00 2022-05-01  1
```

💡 You can specify column list to limit the number of columns and you can pick the columns of your choice from both tables. One special case of concern is when both tables have column with same name then in this case ambiguity (confusion) occurs such that DBMS unable to decide to fetch column from which table. To overcome this ambiguity we have to use table-name along with the column-name. To make the syntax short we can alias the table-name also. Relaxation: When using SELECT * then no need to use table-name.col-name. It will work fine.

An Example

```
Say we have a table 'tblOne'
C1 INT(6)
C2 VARCHAR(4)

Say we have another table 'tblTwo'
C2 VARCHAR(4)
C3 Double(4, 2)

SELECT *
FROM tblOne
INNER JOIN tblTwo
ON tblOne.C2 = tblTwo.C2; #No Error

SELECT C1, C2, C3
FROM tblOne
INNER JOIN tblTwo
ON tblOne.C2 = tblTwo.C2; #Ambuguity; multiple columns C2 are there

SELECT C1, tblOne.C2, C3
FROM tblOne
INNER JOIN tblTwo
ON tblOne.C2 = tblTwo.C2; #No Error

SELECT C1, tblTwo.C2, C3
FROM tblOne
INNER JOIN tblTwo
ON tblOne.C2 = tblTwo.C2; #No Error

SELECT C1, T1.C2, C3
FROM tblOne T1
INNER JOIN tblTwo T2
ON T1.C2 = T2.C2; #No Error

SELECT C1, T2.C2, C3
FROM tblOne T1
INNER JOIN tblTwo T2
ON T1.C2 = T2.C2; #No Error
```

Left Join

Inner Join + All entries of the Left Hand Side table

```
SELECT projection FROM left-table LEFT JOIN right-table
ON left-table.col-name = right-table.col-name;
```

An Example

```
SELECT * FROM category LEFT JOIN product
ON category.cat_id = product.category_id;
```

```
cat_id  cat_name             pro_id  pro_name       MRP       MFG_DATE    category_id
2       Stationary           1       Parker Pen     349.00    2022-06-16  2
3       Furniture            2       Student Chair  1499.00   2021-12-31  3
4       Food Items           3       Dark Chocolate 399.00    2022-04-30  4
1       Electronics          4       Microwave Oven 10500.00  2022-05-15  1
1       Electronics          5       AC             34500.00  2022-05-01  1
5       House-keeping Goods  NULL    NULL           NULL      NULL        NULL
```

Just change the order of tables in above query

```
SELECT * FROM product LEFT JOIN category
ON product.category_id = category.cat_id;

pro_id pro_name          MRP       MFG_DATE    category_id  cat_id      cat_name
4      Microwave Oven    10500.00  2022-05-15  1            1           Electronics
5      AC                34500.00  2022-05-01  1            1           Electronics
1      Parker Pen        349.00    2022-06-16  2            2           Stationary
2      Student Chair     1499.00   2021-12-31  3            3           Furniture
3      Dark Chocolate    399.00    2022-04-30  4            4           Food Items
6      Footware          599.00    2022-05-01  NULL         NULL        NULL
```

Right Join

Inner Join + All entries of the RHS table

```
SELECT projection FROM left-table RIGHT JOIN right-table
ON left-table.col-name = right-table.col-name;
```

An Example

```
SELECT * FROM category
RIGHT JOIN product ON category.cat_id = product.category_id;

cat_id  cat_name             pro_id  pro_name       MRP       MFG_DATE    category_id
1       Electronics          4       Microwave Oven 10500.00  2022-05-15  1
1       Electronics          5       AC             34500.00  2022-05-01  1
2       Stationary           1       Parker Pen     349.00    2022-06-16  2
3       Furniture            2       Student Chair  1499.00   2021-12-31  3
4       Food Items           3       Dark Chocolate 399.00    2022-04-30  4
NULL    NULL                 6       Footware       599.00    2022-05-01  NULL
```

Just change the order of tables in above query

```
SELECT * FROM product RIGHT JOIN category
ON category.cat_id = product.category_id;

pro_id    pro_name         MRP        MFG_DATE    category_id     cat_id  cat_name
4         Microwave Oven   10500.00   2022-05-15  1               1       Electronics
5         AC               34500.00   2022-05-01  1               1       Electronics
```

```
1         Parker Pen      349.00     2022-06-16  2              2       Stationary
2         Student Chair   1499.00    2021-12-31  3              3       Furniture
3         Dark Chocolate  399.00     2022-04-30  4              4       Food Items
NULL      NULL            NULL       NULL        NULL           5       House-keeping Goods
```

💡 ***Tip: You can add additional conditions in join using AND operator***

An Example

```
Query to find cat_name, pro_name, MRP for all product such that include only
those products whose price is more than 1000.00

SELECT cat_name, pro_name, MRP FROM category INNER JOIN product
ON category.cat_id = product.category_id
AND MRP > 1000.00;

OR

SELECT cat_name, pro_name, MRP FROM category
INNER JOIN product ON category.cat_id = product.category_id
WHERE MRP > 1000.00;

cat_name      pro_name         MRP
Furniture     Student Chair    1499.00
Electronics   Microwave Oven   10500.00
Electronics   AC               34500.00
```

Both the queries will produce same result but the former query is having better performance because in the latter query join will be performed then WHERE clause will filter the record so all that will take lot of time than the former syntax. Use of former syntax is preferred because join and conditions are evaluated in a single step.

Tip: Join operation can be applied on any number of tables

An Example

```
customer(cid, cname, dob, address)
account(acc_no, balanace, opening_date, cid)
transaction(tid, amount, transaction_date, acc_no)

Inner join operation on these tables

SELECT C.cid, cname, A.acc_no, balanace, opening_date, transaction_date, amount
FROM customer C INNER JOIN account A
ON C.cid = A.cid
INNER JOIN transaction T
ON T.acc_no = A.acc_no;
```

Self Join

When the primary/unique key of a table is referenced in the table itself by a foreign key column then such foreign key is called self referential foreign key.

For self join operation, we are going to use following table

```
Table-name: soldiers
sol_id sol_name sol_age col_id
S001   ABC      34      NULL
S002   BCD      26      S001
S003   CDE      28      S001
S004   DEF      35      NULL
S005   FGH      26      S004
```

An Example

Write a query to display soldier name along with the colonel name

```
Select S.sol_name "Soldier Name",
       C.sol_name "Colonel Name"
FROM soldier S
INNER JOIN soldier C
ON S.col_id = C.sol_id;

soldier name    colonel name
BCD             ABC
CDE             ABC
FGH             DEF
```

Subqueries

1. Subquery is query inside another query such that the subquery can be written in SELECT, FROM, WHERE and HAVING clause
2. Subquery must be in the parenthesis
3. Subquery is executed first and its result will be used in the outer query.
4. Subquery into the SELECT clause should return only one column

For examples of subqueries, we are using 'st' table again

```
rollNo     name  email        x_per   state
1          ABC   abc@ms.com   77.65   Karnataka
2          BCD   bcd@ms.com   33.00   Tamilnadu
```

```
3          CDE   cde@ms.com   77.24   Maharashtra
4          def   def@ms.com   74.69   West Bengal
5          efg   efg@ms.com   66.00   Kerala
6          fgh   fgh@ms.com   33.00   Delhi
7          ghi   ghi@ms.com   33.00   Punjab
8          ijk   ijk@ms.com   79.36   Haryana
10         AED   NULL         56.00   Karnataka
```

Requirement: You need to write query to find record of students whose percentage is more than the maximum percentage of all scholars from Karnataka

```
SELECT MAX(x_per) FROM st
WHERE state = 'Karnataka';

Result of above query is
MAX(x_per)
77.65

SELECT * FROM st
WHERE x_per > 77.65;

Result of above query is
rollNo     name  email        x_per   state
   8       ijk   ijk@ms.com   79.36   Haryana
```

The above way we have written two queries to reach out to the result, also result of first query we have to remember so this approach is not very suitable because we have to write multiple queries and result has to be remembered so we need a better solution that is to use subquery

Subquery with SELECT clause

Write query to display name, x_per, average percentage for all students, email of all students

```
SELECT name, x_per,
       (SELECT AVG(x_per)
        FROM st) "AVG_PER",
       email
FROM st;

name   x_per   AVG_PER    email
BCD    33.00   58.882222  bcd@ms.com
CDE    77.24   58.882222  cde@ms.com
def    74.69   58.882222  def@ms.com
efg    66.00   58.882222  efg@ms.com
fgh    33.00   58.882222  fgh@ms.com
ghi    33.00   58.882222  ghi@ms.com
ijk    79.36   58.882222  ijk@ms.com
AED    56.00   58.882222  NULL
```

Subquery with FROM clause

When subquery is used with the FROM clause then a temporary table will be generated and this temporary table is not having any name that's why it is mandatory to alias with the result of subquery with FROM clause. This table alias may or may not be used with the column names in outer query.

An Example

```
SELECT rollNo, x_per
FROM (
  SELECT rollNo, name,
  x_per, email
  FROM st
  WHERE x_per > 60.00
) T

OR

SELECT T.rollNo, T.x_per
FROM (
  SELECT rollNo, name,
  x_per, email
  FROM st
  WHERE x_per > 60.00
) T

Result of above query is
rollNo x_per
1      77.65
3      77.24
4      74.69
5      66.00
8      79.36
```

If column aliasing is done in the inner query then aliased columns names has to be use with the outer query.

```
SELECT T.RN, T.XP
FROM (
    SELECT rollNo RN,
           name NM,
           x_per XP,
           email
    FROM st
    WHERE x_per > 60.00
) T

Result of above query is
RN     XP
1      77.65
3      77.24
```

```
4      74.69
5      66.00
8      79.36
```

Subquery with WHERE clause

Query to find record of students whose percentage is maximum among all scholars

```
SELECT *
FROM st
WHERE x_per = MAX(x_per); #Error, because you cannot use aggregate function with WHERE clause

SELECT *
FROM st
WHERE x_per = (
    SELECT MAX(x_per)
    FROM st
);

Result of above query is
rollNo     name  email        x_per   state
   8       ijk   ijk@ms.com   79.36   Haryana
```

Query to find record of students whose percentage is more than the maximum percentage of all scholars from Karnataka

```
SELECT *
FROM st
WHERE x_per > (
    SELECT MAX(x_per)
    FROM st
    WHERE state = 'Karnataka'
);

Result of above query is
rollNo     name  email        x_per   state
   8       ijk   ijk@ms.com   79.36   Haryana
```

Subquery with HAVING clause

Query to find state and average x_per for all state such that include only those states whose average x_per is more than that of Karnataka

```
SELECT state, AVG(x_per)
FROM st
GROUP BY state
HAVING AVG(x_per) > (
  SELECT AVG(x_per)
  FROM st
  WHERE state = 'Karnataka'
)

Result of above query is
state             AVG(x_per)
Haryana           79.360000
Himachal Pradesh  79.360000
Maharashtra       77.240000
West Bengal       74.690000
```

💡 So far we have seen nesting only to one level but nesting can be done at any level

```
SELECT T.rollNo, T.name, T.email
FROM (
  SELECT *
  FROM st
  WHERE x_per >= (
    SELECT x_per
    FROM st
    WHERE state = 'Kerala'
  )
) T

Result of above query is
rollNo    name    email
1         ABC     abc@ms.com
3         CDE     cde@ms.com
4         def     def@ms.com
5         efg     efg@ms.com
8         ijk     ijk@ms.com
11        PQR     pqr@ms.com
```

Subquery with DML statements

Add a record to the product table (used in joins) with following data
pro_id: 7, pro_name: snacks, MRP: 90.00 MFG_DATE: 2022-01-01 category_id: should be same as of category_id of Food Items

```
INSERT INTO product
VALUES
(7, 'snacks', 90.00, '2022-01-01', (SELECT cat_id
                                    FROM category
                                    WHERE cat_name = 'Food Items')
);
```

From the above example it is clear that use of alias is not required if outer query and inner query are applied on the different tables but if they are applied on same table then it is necessary to use alias in the sub query

Add a record to the st table with following data
rollNo: 11, name: PQR, email: pqr@gmail.com, state: Himachal Pradesh, x_per: maximum x_per of all students

```
INSERT INTO st
(rollNo, name, email, state, x_per)
VALUES
(11, 'PQR', 'pqr@gmail.com', 'Himachal Pradesh', (SELECT MAX(x_per)
                                                  FROM st)
);

The above query will produce error because outer query and inner query they are
applicable on the same table so use os alias is mandatory. The correct version
is following-

INSERT INTO st
(rollNo, name, email, state, x_per)
VALUES
(11, 'PQR', 'pqr@gmail.com', 'Himachal Pradesh', (SELECT MAX(x_per)
                                                  FROM st T)
);
```

Single Row Subquery

These subqueries are used to return only one result

The result of Single Row Subquery can be compared using operator like >, <, <=, >=, =, !=, BETWEEN.. AND... etc.

An Example: Query to find record of all scholars whose x_per is more than average x_per of all scholars

```
SELECT *
FROM st
WHERE x_per > (
  SELECT AVG(x_per)
  FROM st
);

Result of above query is
rollNo  name    email       x_per state
1       ABC     abc@ms.com  77.65 Karnataka
3       CDE     cde@ms.com  77.24 Maharashtra
4       def     def@ms.com  74.69 West Bengal
5       efg     efg@ms.com  66.00 Kerala
8       ijk     ijk@ms.com  79.36 Haryana
11      PQR     pqr@ms.com  79.36 Himachal Pradesh
```

Another Example: Query to find record of all scholars whose x_per is more than or equals to maximum percentage of all scholars from delhi and less than or equal to the minimum percentage of all scholars from Kerela

```
SELECT *
FROM st
WHERE x_per BETWEEN (SELECT MAX(x_per)
                     FROM st
                     WHERE state = 'Delhi'
                    )
                    AND
                    (SELECT MIN(x_per)
                     FROM st
                     WHERE state = 'Kerela'
                    );

Result of above query is
rollNo  name    email       x_per state
2       BCD     bcd@ms.com  33.00 Tamilnadu
5       efg     efg@ms.com  66.00 Kerala
6       fgh     fgh@ms.com  33.00 Delhi
7       ghi     ghi@ms.com  33.00 Punjab
10      AED     NULL        56.00 Karnataka
```

Be careful: Query to find record of all scholars whose x_per is not equal to the x_per from the scholars of Karnataka

```
SELECT *
FROM st
WHERE x_per != (SELECT x_per
                FROM st
                WHERE state = 'Karnataka'
);
```

The above query is producing error because the subquery is returning more than one result i.e. the subquery is multi row subquery; multiple results cannot be compared using !=, =, <, >, <=, >=, between ... and.

Multi Row Subquery

These queries are used to return more than one result

The result of Multi Row Subquery can be compared using operator IN, ANY and ALL.

An Example: Query to find record of all scholars whose x_per is not equal to the x_per from the scholars of Karnataka

```
SELECT *
FROM st
WHERE x_per NOT IN (
    SELECT x_per
    FROM st
    WHERE state = 'Karnataka'
);

Result of above query is
rollNo     name  email        x_per   state
2          BCD   bcd@ms.com   33.00   Tamilnadu
3          CDE   cde@ms.com   77.24   Maharashtra
4          def   def@ms.com   74.69   West Bengal
5          efg   efg@ms.com   66.00   Kerala
6          fgh   fgh@ms.com   33.00   Delhi
7          ghi   ghi@ms.com   33.00   Punjab
8          ijk   ijk@ms.com   79.36   Haryana
11         PQR   pqr@ms.com   79.36   Himachal Pradesh
```

An Example of Any: Query to find record of all scholars whose x_per is less than the x_per of any scholar of Karnataka

```
SELECT name, email, x_per
FROM st
WHERE x_per < ANY (
    SELECT x_per
    FROM st
    WHERE state = 'Karnataka'
);

The subquery will give us two results that are 77.65 and 56.00
so the above query is equivalent to

SELECT name, email, x_per
FROM st
WHERE x_per < ANY (77.65, 56.00);

The above query will list all student whose x_per is less than either 77.65 or 56.00
so in short it will return all records whose x_per is less than 77.65
so < ANY means less than the maximum value from the result of subquery.

Result of above query is:
name email      x_per
BCD  bcd@ms.com 33.00
CDE  cde@ms.com 77.24
def  def@ms.com 74.69
efg  efg@ms.com 66.00
fgh  fgh@ms.com 33.00
ghi  ghi@ms.com 33.00
AED  NULL       56.00
```

Another Example of Any: Query to find record of all scholars whose x_per is more than the x_per of any scholar of Karnataka

```
SELECT name, email, x_per
FROM st
WHERE x_per > ANY (
    SELECT x_per
    FROM st
    WHERE state = 'Karnataka'
);

The subquery will give us two results that are 77.65 and 56.00
so the above query is equivalent to

SELECT name, email, x_per
FROM st
WHERE x_per > ANY (77.65, 56.00);

The above query will list all student whose x_per is more than either 77.65 or 56.00
so in short it will return all records whose x_per is more than 56.00
so > ANY means more than the minimum value from the result of subquery.

Result of above query is:
name  email          x_per
ABC   abc@ms.com     77.65
CDE   cde@ms.com     77.24
DEF   def@ms.com     74.69
EFG   efg@ms.com     66.00
IJK   ijk@ms.com     79.36
PQR   pqr@gmail.com  79.36
```

An Example of All: Query to find record of all scholars whose x_per is more than the x_per of all scholar of Karnataka

```
SELECT name, email, x_per
FROM st
WHERE x_per > ALL (
    SELECT x_per
    FROM st
    WHERE state = 'Karnataka'
);

The subquery will give us two results that are 77.65 and 56.00 so the above query
is equivalent to

SELECT name, email, x_per
FROM st
WHERE x_per > ALL (77.65, 56.00);

Result of above query is:
name  email          x_per
IJK   ijk@ms.com     79.36
PQR   pqr@gmail.com  79.36

The above query will list all student whose x_per is more than 77.65 as well
```

```
as 56.00 so in short it will return all records whose x_per is more than 77.65
so > ALL means more than the maximum value from the result of subquery.
```

Another Example of All: Query to find record of all scholars whose x_per is less than the x_per of all scholar of Karnataka

```
SELECT name, email, x_per
FROM st
WHERE x_per < ALL (
    SELECT x_per
    FROM st
    WHERE state = 'Karnataka'
);

The subquery will give us two results that are 77.65 and 56.00
so the above query is equivalent to

SELECT name, email, x_per
FROM st
WHERE x_per < ALL (77.65, 56.00);

The above query will list all student whose x_per is less than 77.65 as well as
56.00 so in short it will return all records whose x_per is less than 56.00
so < ALL means less than the minimum value from the result of subquery.

Result of above query is:
name  email          x_per
BCD   bcd@ms.com     33.00
FGH   fgh@ms.com     33.00
GHI   ghi@ms.com     33.00
```

Transaction Management

Say A has received a cheque of INR 1000/- from B. A presented that cheque to bank and then he thought soon 1000/- will be deducted from B's account and credited to A's account. But suddenly A thought that what happened if 1000/- deducted from B's account and then power failure takes place. Is A right? How DBA will answer this questions? Answer is transaction.

A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit.

In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

A transaction has ACID (Atomicity, Consistency, Isolation, Durability) properties, Take a look at these properties in brief

Atomicity: It refers to the ability of the database to guarantee that either all of the tasks of a transaction are performed or none of them are.

Consistency: It ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not).

Isolation: It refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction.

Durability: Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction.

To start a transaction, use following command

```
START TRANSACTION;
```

start transaction commits he current transaction and start a new transaction. It tells MySql that a new transaction is beginning and statements followed by transaction must be treated as unit. To save changes made during the transaction use following command

```
COMMIT

or

COMMIT WORK
```

Here work keyword is optional.

An Example:
Let say we have a table named as account that have following structure

tid int(10) PRIMARY KEY
name VARCHAR(20) NOT NULL
amount int NOT NULL
email VARCHAR(50) NOT NULL

and make following entries
1, Dinesh, 4000, dinesh@ms.com

2, Jayesh, 4000, jayesh@ms.com

Say we want to transfer INR 1000 from dinesh's account to jayesh's account. To make sure that both operations should be atomic use start transaction and commit

```
START TRANSACTION;
UPDATE account SET amount = (amount - 1000) WHERE tid = 1;
UPDATE account SET amount = (amount + 1000) WHERE tid = 2;
COMMIT;
SELECT * FROM account;

Now result is
tid name    amount  email
1   Dinesh  3000    dinesh@ms.com
2   Jayesh  5000    jayesh@ms.com
```

💡 Tip: Say you forgot to place commit statement after update queries then these queries will not have any effect on database.

During a transaction if any error takes place then the entire transaction will be can be undone using ROLLBACK statement. The ROLLBACK statement cancels the entire transaction and put the database at the beginning point of the transaction. ROLLBACK statement has following syntax

```
ROLLBACK

Or

ROLLBACK WORK
```

Here work keyword is optional. Now execute following queries

```
START TRANSACTION;
UPDATE account SET amount = (amount - 1000) WHERE tid = 1;
UPDATE account SET amount = (amount + 1000) WHERE tid = 2;
ROLLBACK;

SELECT * FROM account;

Now result is

tid name    amount  email
1   Dinesh  3000,   dinesh@ms.com
2   Jayesh  5000,   jayesh@ms.com
```

Because we have ROLLBACK the entire transaction and database again come to the point where it at the starting of transaction.

Sometimes it is not require to cancel the entire transaction it is sufficient to ROLLBACK only a small portion of a transaction, to do so we have to mark SAVEPOINT in transaction. A SAVEPOINT is a marker in a transaction that allow to ROLLBACK a database to the marked point. All changes made to the database after the SAVEPOINT are discarded and changes made prior to the transaction remain unchanged. To insert a SAVEPOINT use statement like

```
SAVEPOINT <savepoint-name>
```

To ROLLBACK to the marked SAVEPOINT use following syntax

```
ROLLBACK TO SAVEPOINT <savepoint-name>
```

Let us again take an example. Again start with table account with following entries in the table. Now execute following set of statements

```
START TRANSACTION;
UPDATE account SET amount = (amount - 500) WHERE tid = 1;
UPDATE account SET amount = (amount + 500) WHERE tid = 2;
SAVEPOINT S1;
UPDATE account SET amount = (amount - 1000) WHERE tid = 1;
UPDATE account SET amount = (amount + 1000) WHERE tid = 2;
ROLLBACK TO S1;
COMMIT;

SELECT * FROM account;

Now result is

tid name    amount  email
1   Dinesh  2500,   dinesh@ms.com
2   Jayesh  5500,   jayesh@ms.com
```

By default AUTOCOMMIT mode is on in MySql, It means all SQL statements are automatically committed by MySql. To To set MySql mode off use following syntax

```
SET AUTOCOMMIT=0;
```

Making AUTOCOMMIT to off work only for a single session. Staring new session automatically set AUTOCOMMIT to on. To explicitly set AUTOCOMMIT mode to on just use following syntax

```
SET AUTOCOMMIT=1;
```

Now Let us take an Example of both, start with the same table accounts. Now execute following SQL queries

```
SET AUTOCOMMIT = 0;
INSERT INTO account (tid, name, amount, email) VALUES (3, 'karan',2000,'karan@ms.com');
ROLLBACK;

SELECT * FROM account;

Now result is

tid name    amount  email
1   Dinesh  2500,   dinesh@ms.com
2   Jayesh  5500,   jayesh@ms.com
```

Yes it is same because AUTOCOMMIT was set to false and ROLLBACK statement put database to initial state. Now try following queries

```
SET AUTOCOMMIT = 1;
INSERT INTO account (tid, name, amount, email) VALUES (3, 'karan',2000,'karan@ms.com');
ROLLBACK;

SELECT * FROM account;

Now result is

tid name    amount  email
1   Dinesh  2500    dinesh@ms.com
2   Jayesh  5500    jayesh@ms.com
3   karan   2000    karan@ms.com
```

Yes this time a new entry is here because AUTOCOMMIT is set to on so all SQL statements are automatically committed.

💡 **Tip**: Making AUTOCOMMIT off has no impact over DDL statements. They are always committed by database automatically.