# Transaction Processing

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Agenda

- What is Transaction
- Transaction Model
- Isolation
  - Classification
- Savepoint and Release point
- Locking
  - Locking Levels
- Views and its types
  - Horizontal
  - Vertical
  - Group
- CHECK OPTION

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# What is Transaction?

- Transaction is a process of executing multiple DML statements sequentially to achieve a final task.
- A transaction which is performing DML operations including Insertion, deletion, update of records ensures *data integrity.*
- In an online application, the transactions happening on the database concurrently ensures the data is accurate and reliable without any loss.
- Usually , in big organizations like banks perform millions of transactions per minute in database environment. All these transactions are processed through DML statements including INSERT, DELETE, UPDATE and SELECT statements.
- All of these transactions ensure the database integrity and accuracy of data for users.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# What is Transaction?

- A transaction with DML statements follows the ACID properties.

- *Atomicity*: It ensures the transaction is *fulfilled*; otherwise *fail* the transaction.

- E.g: Ordering a Pizza is fulfilled with series of transactions like -
  - Choose an item - check the inventory
  - Order the dish - insert order details
  - Complete the order - includes payment transaction.

- If the customer recalls the order at any stage, all the DML operations are rollbacked including changes to inventory, orders and payments.

  However, if the transaction is completed, you will enjoy your Pizza.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

*Consistency* :

Any failure of transactions due to database crash or network failures, the original state of data is retained.

During crash recovery, SQL engine replay the transactions recorded in **doublewrite buffer.** Hence the *data is consistent* in the database without loss of transaction status.

It means it will have a record of the user SQL statements on the database, which are used for recovery process .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

## *Isolation:*

Multiple DML statements concurrently hit the database and isolating one transaction from other transaction is the responsibility of database.

So one individual transaction is isolated from other transactions. It is achieved only by forming the Queue of transactions.

E.g: Assume Withdrawal of amount from ATM & internet transfer are two different transactions occured at the same second. Both transactions are transferring money from the account.

The two individual transactions are isolated , it means the database keeps the transactions in Queue to perform one transaction after the other to ensure the balance is sufficient to transfer the money.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SQL Transaction Model

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

- E.g: A transaction is identified with series of DML operations.

Transaction

- SELECT Operation -1 : Check bank balance
- INSERT Operation -2 : Withdraw money from ATM - new entry into transaction table.
- UPDATE Operation -3 : Update the balance by deducting with drawn money.
- INSERT Operation - 4 : Calculate the service charge on withdrawn amount and insert a new entry in Transaction table

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Start of Transaction

```
START TRANSACTION;

/*     Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

```
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 400000 |

```
/*     Bank charges 0.2% on withdrawn money */
Insert  Transaction values ('4000-1956-2001' ,  -2300.00 * 0.02,   'ATM
Withdrawal' ,  'CA' , now());

/*     Update the old balance with new balance. */
Update ACCOUNT set  balance = balance - 2300 * 0.02  where Acct_Num =
'4000-1956-2001' ;

COMMIT ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Transaction

```
START TRANSACTION;
```

*/* *Check bank balance* */*

```
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

```
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 400000 |

*/* *Withdraw money from ATM* */*

```
Insert Transaction values ('4000-1956-2001' ,  -2300.00,  'ATM Withdrawal' ,
'CA' , now() ) ;
```

```
Insert Transaction values ('4000-1956-2001' ,  -2300.00 ,   'ATM Withdrawal' ,  'CA' , current_date() );

Select * from  Transaction where acct_num  =  '4000-1956-2001'  and tran_amount =  -2300.00
```

Filter Rows: | Export: | Wrap Cell Content:

| Acct_Num | Tran_Amount | Channel | Area | Tran_Date |
|---|---|---|---|---|
| 4000-1956-2001 | -2300.00 | ATM Withdrawal | CA | 2020-09-04 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Transaction

***Before*** Update of ACCOUNT balance.

```
Select * from  ACCOUNT where acct_num  =  '4000-1956-2001'
```

Filter Rows: | Export: | Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | SAVINGS | 400000 | ACTIVE | P |

***Update*** *the old balance with new balance:*

```
UPDATE Account
SET    balance  = balance - 2300
WHERE  Acct_Num = '4000-1956-2001' ;
```

```
Select * from  ACCOUNT where acct_num  =  '4000-1956-2001'
```

Filter Rows: | Export: | Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | SAVINGS | 397700 | ACTIVE | P |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Transaction

*/* Bank charges 0.2% on withdrawn money */*

```
Insert Transaction values
('4000-1956-2001', -2300.00 * 0.02,'ATM Withdrawal','CA' ,current_date());
```

```
Insert Transaction values
('4000-1956-2001' , -2300.00 * 0.02, 'ATM Withdrawal' , 'CA' ,current_date());
Select * from Transaction where acct_num  =  '4000-1956-2001'  and tran_amount =    -46
```

Filter Rows: | Export: | Wrap Cell Content:

| Acct_Num | Tran_Amount | Channel | Area | Tran_Date |
|---|---|---|---|---|
| 4000-1956-2001 | -46.00 | ATM Withdrawal | CA | 2020-05-07 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# End of Transaction

```
/*      Update the old balance with new balance. */

Update ACCOUNT
Set     balance = balance - 2300 * 0.02
WHERE   Acct_Num = '4000-1956-2001' ;

/* End of transaction */
COMMIT;
```

```
Update ACCOUNT set  balance = balance - 2300 * 0.02  where Acct_Num = '4000-1956-2001' ;
Select * from  ACCOUNT where acct_num  =  '4000-1956-2001'  ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | SAVINGS | 397654 | ACTIVE | P |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Transaction Summary

In the above Start and End of transaction,

the transaction is atomic and completes the transaction of ATM withdrawal of money with some charges

- *COMMIT* ensures that transaction is fulfilled

- *ROLLBACK* the transaction, if customer ignores to withdraw the money from ATM, all of the above transactions are reversed from database .

The database will retain your original balance in the account.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Set Transaction

SET transaction controls performs the below tasks

- Encapsulate the DML statements
- Separates the execution of DML statements from DML statements of other transactions.
- Inter- lock the underlying affected records.

Note:

To ensure data integrity, each transaction should be consistent .

It means each DML statement should know the latest value of the underlying record though the record is affected by multiple transactions at same time.

Finally, the data Integrity is achieved only by below:

-Restrict other transactions performing the tasks on same records.

-Maintain the isolation of transactions with other transactions. So that other transactions wait in Queue to perform .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Set Transaction - Example

*Example:*

At 9:00 AM , One transaction tries to update the balance of an Account - 123456 and expected to complete by 9:05 AM.

At 9:02 AM : Meanwhile, another transaction updates the balance of same Account - 123456

Here, the functionality of transactions is:

The second transaction occurred at 9:02 AM will not have access to the Account - 123456 until 9:05 AM.

Hence, it will keep on waiting until the first transaction is completed and execute after 9:05 AM.

This functionality is said to be in ISOLATION which is set by MySQL, and will study more.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION

MySQL - ISOLATION plays an important role in keep waiting of the transactions.
This is done internally and users need not aware of it.

*However, these ISOLATION levels can be* manually altered by user and can modify such constraints and still achieve data integrity and performance.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Classification of Isolation levels

## Classification of ISOLATION levels:

ISOLATION levels are classified as

- REPEATABLE - READ
- READ COMMITTED
- SERIALIZABLE

Note:

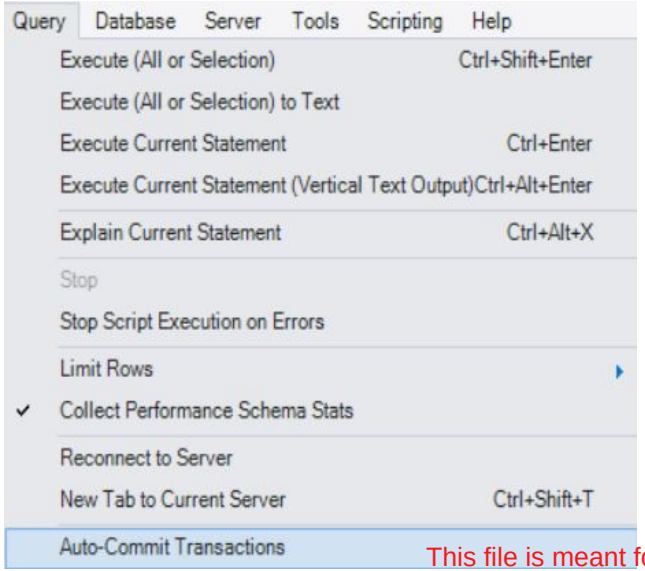
Methods including READ , WRITE on transactions can also achieve Isolation.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Practise Sessions

## Pre-requisite:

- Disable Auto- commit transaction




This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION - Repeatable Read

- REPEATABLE READ: Lock the table in first Transaction affecting other Instances.

Perform below steps in "Instance - 1" and see result of each SQL statement.
( Ensure to *disable* Auto-commit )

/* MYSQL Workbench Instance - 1 */

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

/*      Check bank balance   */
Select Balance from ACCOUNT where Acct_Num =
'4000-1956-2001' ;

/*      Update balance */
Update ACCOUNT set    balance = balance - 2300
where Acct_Num = '4000-1956-2001' ;

/*      Check bank balance   */
Select Balance from ACCOUNT where Acct_Num =
'4000-1956-2001' ;
```

Result of Instance - 1

```
/*     Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;

/*     Update balance */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001' ;

/*     Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 395354 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION - Repeatable Read

- REPEATABLE READ: Experience table level lock using UPDATE statement

/* MYSQL workbench , Instance - 2 */

```
/*      Check bank balance  of different account  */
Select Balance
from ACCOUNT
where Acct_Num = '4000-1956-5698' ;
```

```
/*      Update balance */
Update ACCOUNT
set    balance = balance - 1500
where Acct_Num = '4000-1956-5698' ;
```

Result of Instance - 2

```
Select Balance
from ACCOUNT where Acct_Num = '4000-1956-5698' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 397654 |

```
Update ACCOUNT
set   balance = balance - 1500  where Acct_Num = '4000-1956-5698' ;
```

Output

| Time | Action | Message |
|---|---|---|
| 0:18:18 | INTERRUPT | OK - Query cancelled |
| 0:18:30 | Update ACCOUNT set balance = balance - 1500 where Acct_Num = '4000-1956-5698' | Running... |

- In Instance - 2, the status of UPDATE keeps RUNNING.
- No additional records can be inserted. Because of session - 1 acquired lock at table level.

This file is meant for personal use by shesh.jeppu@gmail.com only
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION - Repeatable Read

- REPEATABLE READ: Experience table level lock using *INSERT statement*

```
/*  MYSQL workbench , Instance -  2 */

/*  Insert new record */

Insert into  ACCOUNT
Values ( 123001 , '4000-1956-9999' , 'SAVINGS',
69000, 'ACTIVE' ,'P' ) ;
```

Result of Instance - 2

```
/*  insert new record */
Insert into  ACCOUNT
Values ( 123001 , '4000-1956-9999' , 'SAVINGS', 69000, 'ACTIVE' ,'P' ) ;
```

Output

| Time | Action | Message |
|---|---|---|
| 00:23:30 | Insert into ACCOUNT Values ( 123001 , '4000-1956-9999' , 'SAVINGS', 69000, 'ACTIVE' ,'P' ) | Error Code: 2013. Lost connec |
| 00:25:33 | Insert into ACCOUNT Values ( 123001 , '4000-1956-9999' , 'SAVINGS', 69000, 'ACTIVE' ,'P' ) | Running... |

- In Instance - 2, the status of INSERT keeps RUNNING.
  Because of session - 1 has acquired the lock on the ACCOUNT table.

This file is meant for personal use by vikesh.ejappa@gmail.com only
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION - Repeatable Read

- REPEATABLE READ: Experience table level lock with below practice, impacts other records.

```
/*  MYSQL workbench , Instance -  2 */

/*      Check bank balance  of different account  */
Select Balance
from ACCOUNT
where Acct_Num = '4000-1956-5698' ;
```

```
/*      Update balance */
Update ACCOUNT
set    balance = balance - 1500
where Acct_Num = '4000-1956-5698' ;
```

Result of Instance - 2

```
Select Balance
from ACCOUNT where Acct_Num = '4000-1956-5698' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 397654 |

```
Update ACCOUNT
set   balance = balance - 1500  where Acct_Num = '4000-1956-5698' ;
```

Output

| Time | Action | Message |
|---|---|---|
| 0:18:18 | INTERRUPT | OK - Query cancelled |
| 0:18:30 | Update ACCOUNT set balance = balance - 1500 where Acct_Num = '4000-1956-5698' | Running... |

- In Instance - 2, the status of UPDATE keeps RUNNING.
- No additional records can be inserted. Because of session - 1 acquired lock at table level.

This file is meant for personal use by shesh.jeppu@gmail.com only
Sharing or publishing the contents in part or full is liable for legal action.

# ISOLATION - Read Committed

- READ COMMITTED : Experience record level lock with below practice.

/* MYSQL workbench - Instance - 1 */

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

/*    Check bank balance   */
Select Balance from ACCOUNT where
Acct_Num = '4000-1956-2001' ;

/*    Update balance */
Update ACCOUNT set
balance = balance - 2300  where
Acct_Num = '4000-1956-2001' ;

/*    Check bank balance   */
Select Balance from ACCOUNT where
Acct_Num = '4000-1956-2001' ;
```

## Result of Instance -1

```
/*    Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;

/*    Update balance */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001' ;

/*    Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 393054 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# READ Committed

- READ COMMITTED : Experience *record level lock only* with below practice.

```
/*  MYSQL workbench Instance - 2 */

/*     Check bank balance  of different account  */
Select Balance
from ACCOUNT
where Acct_Num = '4000-1956-5698' ;


/*  insert a new record */
Insert into ACCOUNT
Values ( '4000-1956-9999' , 'SAVINGS',
69000, 'ACTIVE' ,'P') ;
```

## Result of Instance - 2

```
Select Balance
from ACCOUNT where Acct_Num = '4000-1956-5698' ;
```

Filter Rows: | Export: | Wrap Cell Content:

| Balance |
|---|
| 397654 |

```
/*  insert new record */
Insert into  ACCOUNT
Values ( 123001 , '4000-1956-9999' , 'SAVINGS', 69000, 'ACTIVE' ,'P' ) ;

Select * from Account where acct_num = '4000-1956-9999'
```

Filter Rows: | Export: | Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123001 | 4000-1956-9999 | SAVINGS | 69000 | ACTIVE | P |

- In Instance - 2, the status of INSERT happens successfully.
  However , the Instance -1 acquired **lock at table** level but restricted to existing records only.
  Hence it allowed to insert new records from session -2 .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Transaction with READ and WRITE

Transactions Modifiers:

READ and WRITE can also perform the isolation of transactions in the same Instance .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Set Transaction with default - Read Write

READ WRITE : It is a default transaction,
So any statement is executed in current Instance.

/* Instance - 1 */

```
SET TRANSACTION READ WRITE ;

/*     Check bank balance   */
Select Balance
from ACCOUNT
where Acct_Num = '4000-1956-2001' ;

/*     Update balance */
Update ACCOUNT
set    balance = balance - 2300
where Acct_Num = '4000-1956-2001' ;
```

Result of READ WRITE transaction

```
SET TRANSACTION READ WRITE ;
/*     Check bank balance   */
Select Balance
from ACCOUNT where Acct_Num = '4000-1956-2001' ;
/*     Update balance */
Update ACCOUNT
set   balance = balance - 2300  where Acct_Num = '4000-1956-2001' ;
```

Output

| Time | Action | Message |
|---|---|---|
| 08:17:32 | SET TRANSACTION READ WRITE | 0 row(s) affected |
| 08:17:36 | Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' LIMIT 0, 2000 | 1 row(s) returned |
| 08:17:42 | Update ACCOUNT set balance = balance - 2300 where Acct_Num = '4000-1956-2001' | 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 |

UPDATE statement is executed . Since Read Write is a default transaction modifier.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Set Transaction with READ Only

In the user Instance, transactions are isolated with READ / WRITE / both modifiers.

READ Only :

```
SET TRANSACTION READ ONLY ;

/*    Check bank balance   */
Select Balance
from ACCOUNT
where Acct_Num = '4000-1956-2001' ;

/*    Update balance */
Update ACCOUNT
set    balance = balance - 2300
where Acct_Num = '4000-1956-2001' ;
```

Result of READ only transaction

```
SET TRANSACTION READ ONLY ;

/*    Check bank balance   */
Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' ;

/*    Update balance */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001' ;
```

n Output

| Time | Action | Message |
|---|---|---|
| 08:04:37 | SET TRANSACTION READ ONLY | 0 row(s) affected |
| 08:04:42 | Select Balance from ACCOUNT where Acct_Num = '4000-1956-2001' LIMIT 0, 2000 | 1 row(s) returned |
| 08:04:50 | Update ACCOUNT set  balance = balance - 2300  where Acct_Num = '4000-1956-2001' | Error Code: 1792. Cannot execute statement in a READ ONLY transaction. |

UPDATE statement throws an error :

Cannot execute statement in a READ ONLY transaction.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Altering the Modifiers : Read and Write

Modifiers when set at Instance level can be altered, but it doesn't effect the current transaction.

However it will be effective only from next transaction.

```
SET SESSION TRANSACTION READ WRITE ;

/* In read and write mode */

START Transaction;

SET SESSION TRANSACTION READ ONLY ; /* Effective from next transaction */

        Update ACCOUNT
        set
        balance = balance - 2300
        where Acct_Num = '4000-1956-2001';

/*  DML is executed */

COMMIT  ; /* End the current transaction */
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Altering the Modifiers : Read and Write

## Result

```
SET SESSION TRANSACTION READ WRITE ;
/* In read and write mode */
START Transaction;
SET SESSION TRANSACTION READ ONLY ;     /* Effective only in second transaction */
/*     Accepting DML */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001';
COMMIT;  /* End the transaction */
```

n Output

| Time | Action | Message |
|---|---|---|
| 09:05:33 | commit | 0 row(s) affected |
| 09:05:47 | SET SESSION TRANSACTION READ WRITE | 0 row(s) affected |
| 09:06:23 | START Transaction | 0 row(s) affected |
| 09:06:25 | SET SESSION TRANSACTION READ ONLY | 0 row(s) affected |
| 09:06:30 | Update ACCOUNT set balance = balance - 2300 where Acct_Num = '4000-1956-2001' | 1 row(s) affected Rows matched: 1 Changed: 1 |
| 09:06:38 | COMMIT | 0 row(s) affected |

This file is meant for personal use by lokesh.jojappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Set Transaction

Alter Access modifiers : READ/ WRITE in user SESSION.

/* Second transaction: */

```
START transaction

/*     Update will not execute */
Update ACCOUNT
set
balance = balance - 2300
where Acct_Num = '4000-1956-2001' ;
```

```
SET SESSION TRANSACTION READ WRITE ;
/* In read and write mode */
START Transaction;
SET SESSION TRANSACTION READ ONLY ;     /* Effective only in second transaction */
/*     Accepting DML */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001';
COMMIT;  /* End the transaction */

START transaction ;
/*     Update balance but 1 record effected  */
Update ACCOUNT set   balance = balance - 2300  where Acct_Num = '4000-1956-2001' ;
```

n Output

| Time | Action | Message |
|---|---|---|
| 09:06:25 | SET SESSION TRANSACTION READ ONLY | 0 row(s) affected |
| 09:06:30 | Update ACCOUNT set balance = balance - 2300 where Acct_Num = '4000-1956-2001' | 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 |
| 09:06:38 | COMMIT | 0 row(s) affected |
| 09:27:15 | START transaction | 0 row(s) affected |
| 09:27:31 | Update ACCOUNT set balance = balance - 2300 where Acct_Num = '4000-1956-2001' | Error Code: 1792. Cannot execute statement in a READ ONLY transaction. |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SAVEPOINT and RELEASE SAVEPOINT statement

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SAVEPOINT

- So far, ROLLBACK is issued on entire transaction. SQL engine facilitates to do partially rollback and do final commit.
- SAVEPOINT statement is used to mark a portion of transaction with an identifier in a full length of transaction.
- These checkpoints are later referred by ROLLBACK to undo partial transactions.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SAVEPOINT

- Savepoints are so useful for recovery of errors in database applications.
- Each Savepoint is labelled with a unique name to identify the location of intermediate portion of transaction.

  E.g:

```
  SAVEPOINT Label_1;
SAVEPOINT Label_2;
```

- Using these Labels, user can undo the portion of the transaction.

  E.g:

```
ROLLBACK TO Label_1;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SAVEPOINT

```
SELECT *
FROM   CUSTOMER
WHERE  CUST_ID in
      (123002, 123003 , 123004 )
```

| Cust_Id | Name | Address | State | Phone |
|---|---|---|---|---|
| 123002 | George | 194-6,New brighton | MN | 189761700 |
| 123003 | Harry | 2909-5,walnut creek | CA | 1897617866 |
| 123004 | Jack | 229-5, Concord | CA | 1897627999 |

```
Try:
START TRANSACTION;
SAVEPOINT customer_1;
     DELETE FROM CUSTOMER WHERE
CUST_ID=123004;
SAVEPOINT customer_2;
     DELETE FROM CUSTOMER WHERE
CUST_ID=123003;
SAVEPOINT customer_3;
     DELETE FROM CUSTOMER WHERE
CUST_ID=123002;
```

```
START TRANSACTION;
SAVEPOINT customer_1;
    DELETE FROM CUSTOMER WHERE CUST_ID=123004;
SAVEPOINT customer_2;
    DELETE FROM CUSTOMER WHERE CUST_ID=123003;
SAVEPOINT customer_3;
    DELETE FROM CUSTOMER WHERE CUST_ID=123002;
```

Output

| Time | Action | Message |
|---|---|---|
| 09:54:00 | SAVEPOINT customer_1 | 0 row(s) affected |
| 09:54:00 | DELETE FROM CUSTOMER WHERE CUST_ID=123004 | 1 row(s) affected |
| 09:54:00 | SAVEPOINT customer_2 | 0 row(s) affected |
| 09:54:00 | DELETE FROM CUSTOMER WHERE CUST_ID=123003 | 1 row(s) affected |
| 09:54:00 | SAVEPOINT customer_3 | 0 row(s) affected |
| 09:54:00 | DELETE FROM CUSTOMER WHERE CUST_ID=123002 | 1 row(s) affected |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# SAVEPOINT

```
/* Continue */

ROLLBACK TO customer_2;

COMMIT;
```

```
SELECT *
FROM   CUSTOMER
WHERE  CUST_ID in
       (123002, 123003 , 123004 )
```

| Cust_Id | Name | Address | State | Phone |
|---|---|---|---|---|
| 123002 | George | 194-6,New brighton | MN | 189761700 |
| 123003 | Harry | 2909-5,walnut creek | CA | 1897617866 |

- *Note : Upon commit the transaction, savepoints defined in the transaction are lost.*

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# RELEASE SAVEPOINT

- Savepoints that are created can be deleted in a transaction without affecting the transaction.

- This is called releasing the individual savepoint from the total set of savepoints.

- If SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the earlier SAVEPOINT.

Let us see an example of below:

SELECT * FROM CUSTOMER WHERE CUST_ID in ( 123007, 123006, 123005 )

| Cust_Id | Name | Address | State | Phone |
|---|---|---|---|---|
| 123005 | Jacob | 325-7, Mission Dist | SFO | 1897637000 |
| 123006 | Noah | 275-9, saint-paul | MN | 1897613200 |
| 123007 | Charlie | 125-1, Richfield | MN | 1897617666 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# RELEASE SAVEPOINT

Example:

```
START TRANSACTION;

SAVEPOINT customer_1;
     DELETE
     FROM CUSTOMER
     WHERE CUST_ID =123007;

SAVEPOINT customer_2;
     DELETE
     FROM CUSTOMER
     WHERE CUST_ID =123006;

SAVEPOINT customer_3;
     DELETE
     FROM CUSTOMER
     WHERE Cust_Id=123005;

RELEASE SAVEPOINT customer_2;

ROLLBACK  to customer_2 ;
ROLLBACK  to customer_3 ;
```

## Result

```
START TRANSACTION;
SAVEPOINT customer_1;
    DELETE FROM CUSTOMER WHERE CUST_ID =123007;
SAVEPOINT customer_2;
    DELETE FROM CUSTOMER WHERE CUST_ID =123006;
SAVEPOINT customer_3;
    DELETE FROM CUSTOMER WHERE Cust_Id =123005;
RELEASE SAVEPOINT customer_2;
ROLLBACK TO customer_2;
ROLLBACK TO customer_3;
```

Output

| Time | Action | Message |
|---|---|---|
| 14:46:28 | DELETE FROM CUSTOMER WHERE CUST_ID =123007 | 1 row(s) affected |
| 14:46:28 | SAVEPOINT customer_2 | 0 row(s) affected |
| 14:46:28 | DELETE FROM CUSTOMER WHERE CUST_ID =123006 | 1 row(s) affected |
| 14:46:28 | SAVEPOINT customer_3 | 0 row(s) affected |
| 14:46:28 | DELETE FROM CUSTOMER WHERE Cust_Id =123005 | 1 row(s) affected |
| 14:46:28 | RELEASE SAVEPOINT customer_2 | 0 row(s) affected |
| 14:46:28 | ROLLBACK TO customer_2 | Error Code: 1305. SAVEPOINT customer_2 does not exist |
| 14:46:42 | ROLLBACK TO customer_3 | Error Code: 1305. SAVEPOINT customer_3 does not exist |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# RELEASE SAVEPOINT

Rollback issued before RELEASE will be successful.

E.g: **ROLLBACK** TO *Customer_1* ;

```
START TRANSACTION;
SAVEPOINT customer_1;
    DELETE FROM CUSTOMER WHERE CUST_ID =123007;
SAVEPOINT customer_2;
    DELETE FROM CUSTOMER WHERE CUST_ID =123006;
SAVEPOINT customer_3;
    DELETE FROM CUSTOMER WHERE Cust_Id =123005;
RELEASE SAVEPOINT customer_2;
ROLLBACK TO customer_2;
ROLLBACK TO customer_3;
ROLLBACK TO customer_1 ;
```

Output

| Time | Action | Message |
|---|---|---|
| 14:46:28 | SAVEPOINT customer_2 | 0 row(s) affected |
| 14:46:28 | DELETE FROM CUSTOMER WHERE CUST_ID =123006 | 1 row(s) affected |
| 14:46:28 | SAVEPOINT customer_3 | 0 row(s) affected |
| 14:46:28 | DELETE FROM CUSTOMER WHERE Cust_Id =123005 | 1 row(s) affected |
| 14:46:28 | RELEASE SAVEPOINT customer_2 | 0 row(s) affected |
| 14:46:28 | ROLLBACK TO customer_2 | Error Code: 1305. SAVEPOINT customer_2 does not exist |
| 14:46:42 | ROLLBACK TO customer_3 | Error Code: 1305. SAVEPOINT customer_3 does not exist |
| 14:50:33 | ROLLBACK TO customer_1 | 0 row(s) affected |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# COMMIT Statement

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# COMMIT

- COMMIT ends the current transaction.
- It permanently  saves the transaction effected changes to the physical data files and make others visible to the data changes.
- After the COMMIT has been issued, the current transaction will exit and savepoint will disappear.
- There is no option of database that will "undo" the transaction after committing the data.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# COMMIT

**Example:**

```
SELECT *
FROM ACCOUNT
WHERE Acct_status = 'INACTIVE'
AND  (balance = 0 or balance is NULL  ) ;
```

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123007 | 9000-1700-7777-4321 | CREDITCARD | 0 | INACTIVE | P |
| 123008 | 5000-1700-7755 | SAVINGS | NULL | INACTIVE | P |

```
DELETE
FROM ACCOUNT
WHERE Acct_status = 'INACTIVE'
AND  (balance = 0 or balance is NULL);
```

**COMMIT;**

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ROLLBACK

- In a current running transaction which is not yet physically committed to database, the transactional changes can be rollbacked and retain the original status of the data.

- ROLLBACK plays a great roll during interactive operations when user tries to restrain from modifying the data.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ROLLBACK

Delete few rows and try to ROLLBACK to retain the original data.

```
DELETE FROM CUSTOMER  WHERE CUST_ID =123006

select * from CUSTOMER;
```

| CUST_ID | NAME | ADDRESS | STATE_CODE | TELEPHONE |
|---|---|---|---|---|
| 123001 | Oliver | 225-5, Emeryville | 452 | 1897614500 |
| 123002 | George | 194-6,New brighton | 875 | 189761700 |
| 123003 | Harry | 2909-5,walnut creek | 232 | 1897617866 |
| 123004 | Jack | 229-5, Concord | 501 | 1897627999 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# ROLLBACK

Performing ROLLBACK

```
ROLLBACK;

select * from CUSTOMER;
```

| CUST_ID | NAME | ADDRESS | STATE_CODE | TELEPHONE |
|---|---|---|---|---|
| 123006 | Kiran | | 532 | 9542434422 |
| 123006 | Kiran | | 532 | 9542434422 |
| 123001 | Oliver | 225-5, Emeryville | 452 | 1897614500 |
| 123002 | George | 194-6,New brighton | 875 | 189761700 |
| 123003 | Harry | 2909-5,walnut creek | 232 | 1897617866 |
| 123004 | Jack | 229-5, Concord | 501 | 1897627999 |

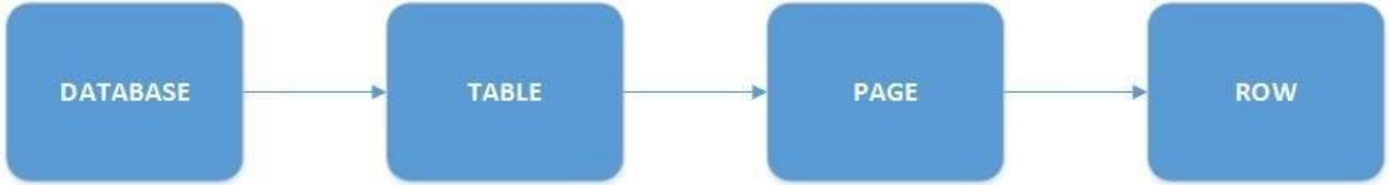
This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Locking

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Lock

- LOCKS are acquired by user transaction in a session.
- SQL Engine locks  objects when the transaction begins the DML statements on each record.
- To ensure data integrity, DML statements acquire explicit locks on data records or tables.
- Multiple users in an integrated environment can see changes applied by each other.
- Afte
- Lock
  The




This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Locking Levels

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Locking Levels

- Several types of locks are used for concurrency that each lock is specific to a transaction and does not affect other transaction.

- Below are the system resources on which Locks are acquired.

- RID: (Row ID) : Row-id is a unique address of individual record in a table.
  DML statements when executed will try to isolate the records from other transactions using this rowid.Hence the row-ids are locked by DML statements, and then perform the operations.

- Table: A transaction with DML statements lock the entire table and corresponding indexes.
  This locks are more reliable in batch processing when performing huge volume of data, and restricts users to access the tables. So that the performance will not be degraded in terms of taking snapshots of data.

- Key: Indexes are generated by default when primary Key is created on a table. Hence the locks are acquired on these keys .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Locking Levels

*Memory level locks : Database memory is allocated with pages and pages forms an extent. Locks do occur at these stages.*

Page: A page consists of 8-kilobyte (KB) used by a data file or index . Acquire the locks at page level and extent level.

Extent: Similarly an extents can be locked which consists of contiguous 8 data pages.

Database: Entire Database can be locked for maintenance purpose .

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared and Exclusive Locks

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared and Exclusive Locks

- Every transaction executes DML statements needs to acquire *Row-level locking*.
- Here are the two types.
  - Shared lock or Read Lock
  - Exclusive lock or Write Lock

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared Locks

- Shared lock is an initial lock acquired by a SELECT statement of a transaction to lock the rows for applying next DML statements.

- Shared lock is  useful to avoid deadlocks of multiple transactions if they are using same DML statements.

- However, the other transactions can also acquire multiple shared locks (S) on the same records locked by first transaction.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared Locks between two SELECT Queries

For example: Consider the following two sessions

*SHARED lock is acquired by two sessions without any error.*

**Instance - 1**

```
START TRANSACTION;

SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001'
LOCK IN SHARE MODE
```

```
START TRANSACTION;

SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
```

Filter Rows: Export: Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | SAVINGS | 383854 | ACTIVE | P |

Output

| Time | Action | Message |
|---|---|---|
| 15:28:54 | START TRANSACTION | 0 row(s) affected |
| 15:28:54 | SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LIMIT 0, 2000 LOCK IN SHARE MODE | 1 row(s) returned |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared Locks between two SELECT queries

Shared lock is also acquired by Instance -2 .

**Instance - 2**

```
START TRANSACTION;

SELECT *
FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001'
LOCK IN SHARE MODE
```

```
START TRANSACTION;
SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
```

Filter Rows: Export: Wrap Cell Content:

| Cust_Id | Acct_Num | Acct_Type | Balance | Acct_status | Relation |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | SAVINGS | 383854 | ACTIVE | P |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared Locks with Update statement

**Session - 1**

```
START TRANSACTION;

SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001'
LOCK IN SHARE MODE;

UPDATE ACCOUNT
Set balance =
balance -  balance * 0.04
WHERE Acct_Num = '4000-1956-2001'
```

```
START TRANSACTION;
SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE;
/* Update after */
UPDATE ACCOUNT
Set balance = balance -  balance * 0.04
WHERE Acct_Num = '4000-1956-2001'
```

Output

| Time | Action | Message |
|---|---|---|
| 15:37:21 | START TRANSACTION | 0 row(s) affected |
| 15:37:21 | SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LIMIT 0, 2000 LOCK IN SHARE MODE | 1 row(s) returned |
| 15:37:33 | UPDATE ACCOUNT Set balance = balance - balance * 0.04 WHERE Acct_Num = '4000-1956-2001' | 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Shared Locks

**Session - 2**

```
START TRANSACTION;

SELECT *
FROM ACCOUNT
WHERE
Acct_Num = '4000-1956-2001'
LOCK IN SHARE MODE
```

```
START TRANSACTION;

SELECT * FROM ACCOUNT
WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
```

Output

| Time | Action | Message |
|---|---|---|
| 15:39:52 | START TRANSACTION | 0 row(s) affected |
| 15:39:52 | SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LIMIT 0, 2000 LOCK IN SHARE MODE | Running... |

*Running / Awaiting :*

Since the DML statement is executed in Instance -1 for same record, a shared lock is occured in Instance - 2 .

Instance - 2 is in Queue and gains the lock on the record after instance - 1 activity is done.

This file is meant for personal use by lokesh.lejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Exclusive (X) Locks

- Exclusive locks are acquired by DML statements: DELETE, INSERT and UPDATE.
- Data cannot be modified at a time by two DML statements. If they then its a deadlock or one transaction keeps waiting for other transaction.
- Hence this lock is mandatorily acquired by any DML statement by default when executed to modify the records.
- This lock cannot be shared with any other transactions as they have to keep waiting until the first DML transaction completes and commits its changes to database.
- Note: Once the Exclusive lock ( X) is acquired , other transactions gain neither of the Shared ( S) nor Exclusive (X) Locks

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Exclusive (X) Locks

**Session - 1** /* acquired shared lock */

```
START TRANSACTION
SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
```

**Session - 2** /* acquired shared lock */

```
START TRANSACTION
SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
```

**Session - 1** /* acquired shared lock **and exclusive lock** */

```
START TRANSACTION
SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
UPDATE ACCOUNT Set balance = balance -  balance * 0.04 WHERE Acct_Num = '4000-1956-2001'
```

**Session - 2** /* acquired shared lock **but did not gain exclusive lock ( waiting )** */

```
START TRANSACTION
SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
UPDATE ACCOUNT Set balance = balance -  balance * 0.04 WHERE Acct_Num = '4000-1956-2001'
```

**Session - 3** /* **Neither acquires Shared lock(S) nor Exclusive ( X) lock** */

```
START TRANSACTION
SELECT * FROM ACCOUNT WHERE Acct_Num = '4000-1956-2001' LOCK IN SHARE MODE
UPDATE ACCOUNT Set balance = balance -  balance * 0.04 WHERE Acct_Num = '4000-1956-2001'
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Virtual Tables

- A virtual table is a *derived form of data* from a physical database tables.
- Virtual tables are logically represented using physical data but *do not store* any data.
- A virtual table is a construct of record set that are not actually tables.
  so that these record sets are referenced in SQL statements like normal tables.
- Virtual tables otherwise called as views and are stored as database objects in MYSQL  Information_schema, and can be retrieved without re-writing it again.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# What is a view?

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# What is view?

- A view is a database object that is created using a Select Query with complex logic.
- So views are said to be a logical representation of physical data
- Views behave like a physical table and users can use them as database objects in any part of SQL queries
- DML operations can be performed on base tables via views.
- Since they are stored in the data dictionary , they can be retrieved easily

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Types of views: Simple View

- Simple views are created with a select query written using a single table
- DML operations are easily performed just like performing on tables

```
Create VIEW Simple_view as
Select * from CUSTOMER ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Complex View:

- Complex view is created using a Select query written with multiple tables using JOINS, sub-queries, with clauses and conditionally filtered

```
Create VIEW Complex_view as
SELECT  c.Cust_Id ,
        a.ACCOUNT
From CUSTOMER c
JOIN ACCOUNT  a
Where c.Cust_Id = a.Cust_Id
And   a.balance > 300000
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Inline View:

- A subquery is also called as an inline view  if and only if it is called in FROM clause of a SELECT query

- It is not stored in any data dictionary like other views. It is dynamically defined in queries

```
SELECT * FROM
( SELECT  c.Cust_Id , ba.ACCOUNT
  From    CUSTOMER bc
JOIN ACCOUNT   ba
Where c.Cust_Id = ba.Cust_Id
And ba.balance > 300000)
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized View:

- Unlike views, Materialized views stores the results of a complex SQL queries
- These are so useful when the SQL engine takes time in execution of the SQL query logic and retrieve the results
- Materialized views are also called as Snapshots of database tables

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized View:

- These views are refreshed with new data with scheduled intervals
- Materialized views are mostly used in data marts where there is a need for handling large volume of data that refers to history data

Data marts is a repositories of summarized data. This data is collected for analysis on a specific section or unit within an organization, for example, the sales department.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized View:

```
create MATERIALIZED VIEW M_VW as SELECT  c.Cust_Id ,
ba.ACCOUNT
From CUSTOMER c
JOIN ACCOUNT   ba
Where bc.Cust_Id = ba.Cust_Id
And  ba.balance > 99999
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Refresh Methods

- Fast & Complete
- A fast refresh uses a log that compares the history and recent changes to its database tables and captures only the changes
- Complete refresh truncates the whole materialized view data and refreshes with new changed data
- Complete refresh is slow when compared with fast refresh method

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How the RDBMS handles views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- Views are stored in data dictionary. It means once they are created , they are used by several users at a time  in different sessions

```
Select * from ALL_VIEWS where view_name in ('SIMPLE_VIEW'
, 'COMPLX_VIEW')
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- The underlying data of base tables dynamically reflects in views since it logically represents the physical data
- However, any updates to data is visible to views only when it is committed
- E.g. Open two different SQL sessions and test Simple_View
- Step1: In one session, issue the below:

```
Select * from Simple_view where Cust_Id = 123001
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- Step2: In another session, issue the below ( no auto commit/ commit )
- Update CUSTOMER set telephone = 9701499878 where Cust_Id = 123001
- Step3: Come back to first session , and again check the view to see the data doesn't reflects the updated telephone

```
Select * from Simple_view where Cust_Id = 123001
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- Step4: Now go back to second session , and issue commit on Update DML statement

  Commit;

- Step5: Once again come back to first session, and check the updated telephone number.

```
Select * from Simple_view where Cust_Id = 123001
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- Views are invoked in FROM clauses of SELECT statements and avoids the complexity of debugging and writing large code

```
E.g:
Select  CA_vw.Cust_Id,
        bat.Acct_Num,
        bat.Tran_Amount,
        CA_vw.balance
from
Transaction bat
JOIN
Complex_view  CA_vw
ON bat.Acct_Num = CA_vw.Acct_Num
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- In the previous example, the query looks so simple as *Complex_view* has taken care of additional logic of conditions and joining CUSTOMER table, and avoided too much of parent Query logic.

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- Views avoids direct access to physical data and protects from unauthorized users
- Ex: If any user fires a SELECT query directly on physical table while the data is being loaded into base tables at the same time
- Here RDBMS takes a **before** and **after** images of table data each time when updating is happening to data

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# How RDBMS handles views

- The after image data is then stored to physical disk
- During this time, if any user tries to do a SELECT on physical data , it will show the data from **before** image since the data is not commited
- Before images are however deleted once the after image data is committed
- Hence there is a lot of overhead to system to maintain **before** images for multiple users to ensure data integrity

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Advantages and Disadvantages of views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Advantages:

- Views are stored queries and re-used
- They are accessible by multiple users in different sessions
- Scalability is high
- They hide the confidential columns data like SSN, date of birth, Address , telephone

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Advantages:

- Avoids direct access to physical data
- Views protects the data when views are invoked by downstream users to build web applications that are prone to hackers
- Materialized views are useful in a centralized environment where multiple users access it in batches of data rather streaming data

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Disadvantages:

- When a base table of an associated view is dropped, it becomes obsolete
- Queries using views are bit slow compared to accessing data directly from physical tables
- Because , SQL engine will take an additional processing of semantics checking of view each time when queries are running on it

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Disadvantages:

- Views created using aggregate functions are restricted to use in WHERE clauses
- Materialized views stores physical data hence it holds redundant data and results in consumption of additional space for its Mviews and Logs
- Views are in-efficient when joined with remote database tables as they consume lot of I/O transactions between local and remote database tables

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Horizontal View

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Horizontal View

- A Horizontal view dynamically represents the data without aware of the columns
- The view does not necessary to know about the number of columns and its data types of the respective base tables
- However the base tables are explicitly defined in the view

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Example

```
Create view Horizontal_view as
Select ba.*
From ACCOUNT  where balance > 99999
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Example

- In this previous example, the wildcard - "*" references all of the columns belongs to base tables mentioned in the view

- Any changes to the column names , data types doesn't affect the view creation

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Vertical View

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Vertical View

- A vertical view built with  predefined columns , however the changes to column data types does not affect the view unless they are type-casted or applied with any system functions.

```
CREATE VIEW Vertical_view AS
SELECT Cust_Id,
       Acct_Num,
       Balance,
       Acct_Type,
       Acct_status,
       Relation
 FROM ACCOUNT
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Vertical View

- In this example, the columns that are explicitly mentioned while creating the view and those are same as base tables
- Otherwise it throws an error if any changes applied to columns

| customer_id | account_number | balance_amount | account_type | Account_status | Relation_ship |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | 400000 | SAVINGS | ACTIVE | P |
| 123003 | 4000-1956-2900 | 750000 | SAVINGS | INACTIVE | P |
| 123004 | 4000-1956-3401 | 655000 | SAVINGS | ACTIVE | P |
| 123001 | 4000-1956-3456 | 200000 | SAVINGS | ACTIVE | P |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Horizontal view Vs Vertical view

| Horizontal views | Vertical views |
| --- | --- |
| Columns are represented with “*” card | Columns are explicitly defined |
| Query execution is dynamic and re-calculate the execution plan each time because columns are unknown. | Underlying query execution plan is not repeated because view columns are in sync with base tables. |
| Maintenance is easy as the view is not worried about changes to columns | View maintenance is required and need to be adjusted according to base table changes. |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Vertical View

| Horizontal views | Vertical views |
|---|---|
| Users have access to all view columns like accessing all base table columns. | Many of the columns are hidden. |
| Users can apply any data type conversions while using the views. | Any data type conversions can be applied unless the other functions inside view definition are not compatible. |
| Views can be updated easily. | These views can also be updated unless any functions are used on columns in view. |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Row Column subset view

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Row column subset View

- A View can be created to represent the subset of records and lesser columns in order to filter the records and hide the columns

```
CREATE VIEW Subset_view AS
SELECT
      Cust_Id,
      Acct_Num,
      Balance
FROM  ACCOUNT
Where Balance > 700000 ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Row column subset View

- In this example, only few columns are displayed that are defined in the subset view. The other columns are hidden

- Also, the records are filtered for which the balance is less than or equal to 700000

| customer_id | account_number | balance_amount | account_type | Account_status | Relation_ship |
|---|---|---|---|---|---|
| 123002 | 4000-1956-2001 | 400000 | SAVINGS | ACTIVE | P |
| 123003 | 4000-1956-2900 | 750000 | SAVINGS | INACTIVE | P |
| 123004 | 4000-1956-3401 | 655000 | SAVINGS | ACTIVE | P |
| 123001 | 4000-1956-3456 | 200000 | SAVINGS | ACTIVE | P |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Grouped Views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Group View

- Grouping views are created using a SELECT query with grouping functions applied on multiple rows of data
- The grouping functions are dynamically applied on rows of data and displays aggregate results only

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Group View

```
Create view Grouping_view
(Acct_Num, total_transaction)
as
select  Acct_Num ,
        sum(Tran_Amount )
from Transaction
group by Acct_Num  ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Group View

| account_number | total_transaction |
|---|---|
| 4000-1956-2001 | 8400.00 |
| 4000-1956-3401 | 8000.00 |
| 4000-1956-3456 | -2000.00 |
| 4000-1956-5102 | -6500.00 |
| 4000-1956-5698 | 9000.00 |
| 4000-1956-9977 | 229000.00 |
| 5000-1700-6091 | 40000.00 |
| 5000-1700-7791 | 40000.00 |
| 5800-1700-9800-7755 | -18000.00 |
| 5890-1970-7706-8912 | -26000.00 |
| 5900-1900-9877-5543 | -20000.00 |
| 9000-1700-7777-4321 | -15500.00 |

In this example, only the aggregate transaction amount is displayed

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Joined Views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Joined View

- These views represent the data from more than one base tables joined using key columns
- The underlying query can have JOINS or subqueries on base tables

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Joined View

```
Create View Joined_view
As
Select c.Cust_Id,
       c.Name,
       a.Acct_Num,
       a.Acct_Type,
       a.Balance,
       c.telephone
FROM ACCOUNT  a
JOIN CUSTOMER c
ON a.Cust_Id = c.Cust_Id ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Joined View

| customer_id | customer_name | account_number | account_type | balance_amount | telephone |
|---|---|---|---|---|---|
| 123002 | George | 4000-1956-2001 | SAVINGS | 400000 | 1897617000 |
| 123003 | Harry | 4000-1956-2900 | SAVINGS | 750000 | 1897617866 |
| 123004 | Jack | 4000-1956-3401 | SAVINGS | 655000 | 1897627999 |
| 123001 | Oliver | 4000-1956-3456 | SAVINGS | 200000 | 1897614500 |
| 123005 | Jacob | 4000-1956-5102 | SAVINGS | 300000 | 1897637000 |

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Joined View

- In this example, the underlying query is joined with two tables and selected their respective columns
- From the view point, the representation of columns are viewed as single entity without any complexity

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# CHECK OPTION

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION

- Simple views can be queried and allow DML operations
- In order to restrict users to perform DML operations on sensitive columns, WITH CHECK OPTION" is optionally used as a constraint

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Example

```
Create view View_with_Check
As Select * From ACCOUNT
Where Balance > 5000
WITH CHECK OPTION ;

Insert into View_with_Check values
( 123001, '4000-7843-3002', 'SAVINGS',4000,'ACTIVE',
'P')
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Example

```
#1369 - CHECK OPTION failed 'tantu.view_with_check'
```

- In this example, you can insert any balance amount without the CHECK OPTION constraint irrespective of the condition in the view

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( LOCAL)

- LOCAL CHECK option pertains to only local conditions of the view and ignores the conditions of dependent views

```
Create view normal_view as
Select * from ACCOUNT
where Balance >= 80000;

create view local_view as
select * from normal_view
where Balance  <= 90000
with local check option;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( LOCAL)

```
Insert into cascaded_view values
( 123001, '6500-3823-4032', 'SAVINGS',65000,
'ACTIVE','P')
```

```
#1369 - CHECK OPTION failed 'tantu.view_with_check'
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( LOCAL)

- In this first example, you can bypass the condition and insert balance amount of 85000 in normal_view
- Local_view is created on normal_view will be local will validate its own conditions
- Hence the records with any balance less than 90k are accepted, and ignores the  dependent view condition

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( CASCADED)

- A normal view conditions are bypassed when performing DML statements
- CASCADED CHECK option ensures to satisfy all of the conditions of dependent views

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( CASCADED)

```
Create view normal_view as
Select * from ACCOUNT
where Balance >= 80000;

create view cascaded_view as
select * from normal_view
where Balance  =< 90000
with cascaded check option;

Insert into cascaded_view values
(123001, '5000-1253-4312', 'SAVINGS' , 70000,
'ACTIVE','P'
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# WITH CHECK OPTION ( CASCADED)

```
#1369 - CHECK OPTION failed 'tantu.view_with_check'
```

- In this first example, you can bypass the condition and insert balance amount of 85000 in normal_view
- Such cases, cascaded_view is created on normal_view to ensure that the  conditions of both views are satisfied
- Hence the records with balance between 80k and 90k are only accepted

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Drop view

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Drop View

- Without affecting the base table data, any type of view can be dropped

  E.g:

  Drop view view_name;

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized view

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized views ( Oracle )

- M-views are logical created using physical tables but it stores the data when they are executed
- Materialized views are executed frequently to refreshed with new data of base tables
- M-views uses mainly two refresh methods : Fast and Complete

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized views ( Oracle )

```
Create materialized view mview_account
As Select * From ACCOUNT
Where Balance > 500000 ;
```

- In this example, materialized view is created with a SQL query logic but does not hold any physical data
- In order to feed the data to materialized views, materialized views must be refreshed

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized views ( Oracle )

- Materialized views are refreshed with physical data. Let us look at how they are refreshed

  E.g:

  execute dbms_mview.refresh( '*mview_account*') ;

```
Select  * from mview_account ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Materialized views ( Oracle )

| Customer_id | Account_Number | Account_type | Balance_amount | Account_status | Relation_ship |
|---|---|---|---|---|---|
| 123003 | 4000-1956-2900 | SAVINGS | 750000 | INACTIVE | P |
| 123004 | 4000-1956-3401 | SAVINGS | 655000 | ACTIVE | P |
| 123007 | 4000-1956-9977 | RECURRING DEPOSITS | 7025000 | ACTIVE | S |
| 123001 | 5000-1700-3456 | RECURRING DEPOSITS | 9400000 | ACTIVE | S |
| 123002 | 5000-1700-5001 | RECURRING DEPOSITS | 7500000 | ACTIVE | S |
| 123004 | 5000-1700-6091 | RECURRING DEPOSITS | 7500000 | ACTIVE | S |

- Mview_account is refreshed with data only after it is explicitly refreshed with system package

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Complete Refresh:

- Materialized view uses complete refresh method to truncate and refresh with new data from Query it is defined with

```
Create materialized view
REFRESH COMPLETE
mview_account_RC
As Select * From ACCOUNT
Where Balance > 500000
```

execute dbms_mview.refresh( list => ‘mview_account_RC’ , method => 'C’);

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Complete Refresh:

- In this example, materialized view always uses complete method to reload the data every time it is refreshed explicitly
- Here the refresh time is equals to the time of query execution and loading of the data

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# FAST Refresh:

- Materialized view uses FAST refresh method with only changed data of the base query

```
Create materialized view fastrefresh_log on ACCOUNT
with sequence ;

Create materialized view
REFRESH FAST
mview_account_RFast
As Select * From ACCOUNT
Where Balance > 500000 ;
```

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# FAST Refresh:

```
execute dbms_mview.refresh( list => '
mview_account_RFast' ,  method => 'F');
```

- In this example, materialized view always uses FAST method to capture only updated or inserted data every time it is refreshed explicitly
- The changed data is always maintained in a log created on the base table which being used by fast refresh later

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

# Thank You

This file is meant for personal use by lokesh.jejappa@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.