UNIT-1

TRANSACTION PROCESSING CONCEPT

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to performoperations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)

2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

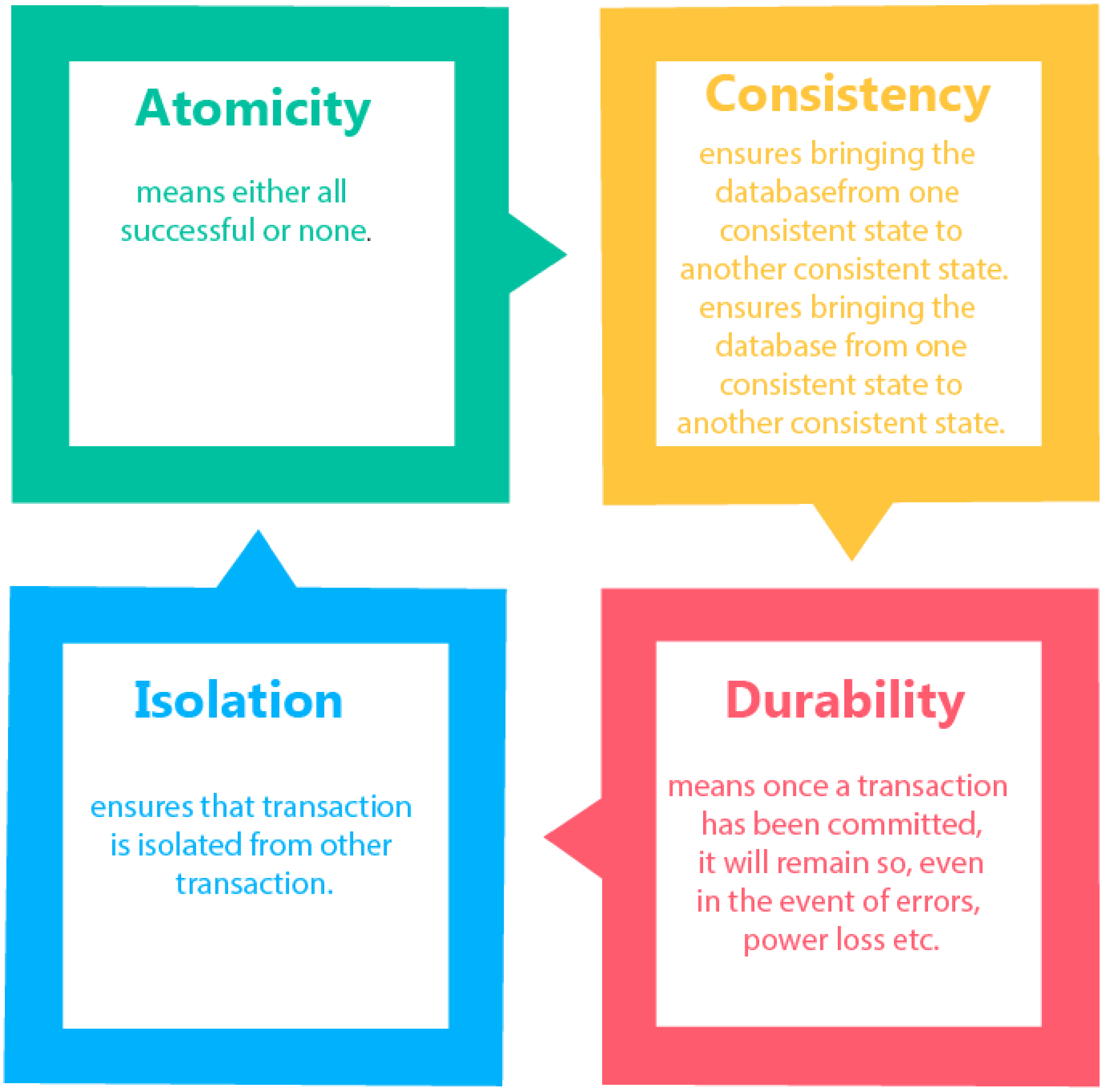
Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction isaborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treatedas one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and afterthe transaction.
- The execution of a transaction will leave a database in either its prior stable state or a newstable state.
- The consistent property of database states that every transaction sees a consistent databaseinstance.
- The transaction is used to transform the database from one consistent state to anotherconsistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be usedby the second transaction until the first one is completed.

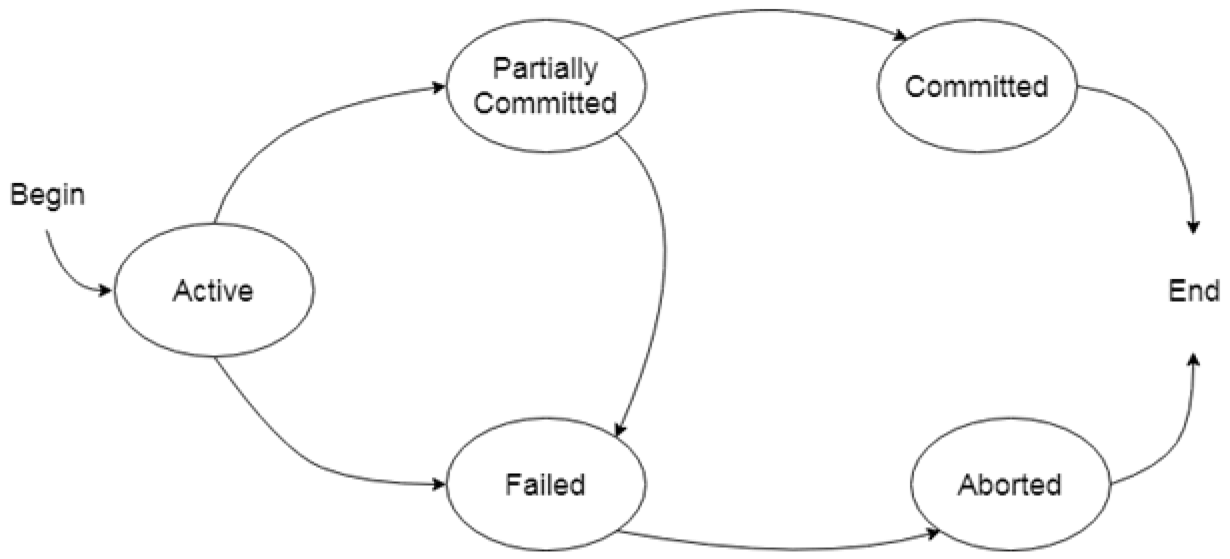
- In isolation, if the transaction T1 is being executed and using the data item X, then that dataitem can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistentstate. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is beingexecuted.
- For example: Insertion or deletion or updating a record is done here. But all the records arestill not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data isstill not saved to the database.

- In the total mark calculation example, a final display of the total marks step is executed inthis state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

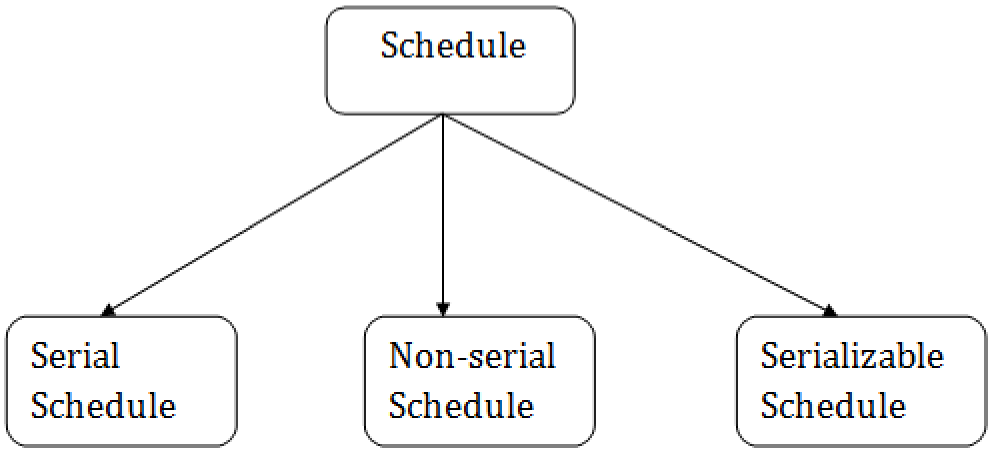
- If any of the checks made by the database recovery system fails, then the transaction is saidto be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetchthe marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If notthen it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction,all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the twooperations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

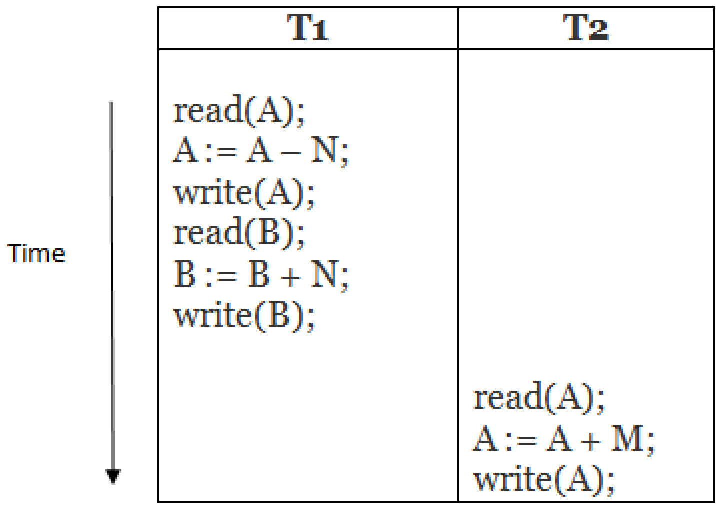
- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operationsof the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. Ithas interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transactionto execute concurrently without interfering with one another.

- It identifies which schedules are correct when executions of the transaction have interleavingof their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactionsexecuted serially.

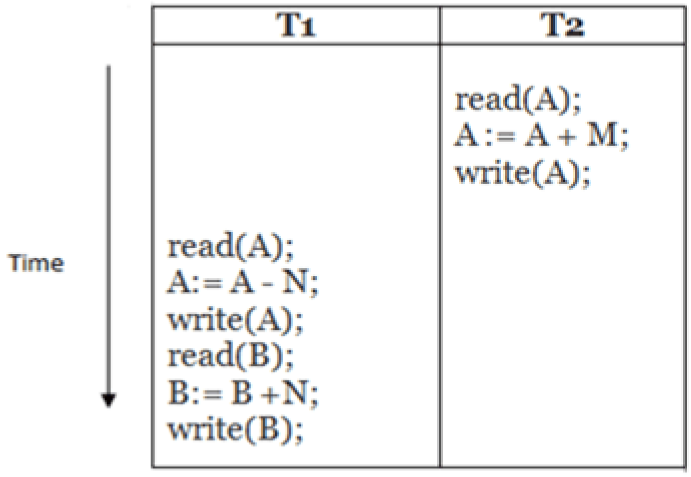
(a)



T1	T2
read(A);	
A := A – N;	
write(A);	
read(B);	
B := B + N;	
write(B);	
	read(A);
	A := A + M;
	write(A);

Schedule A

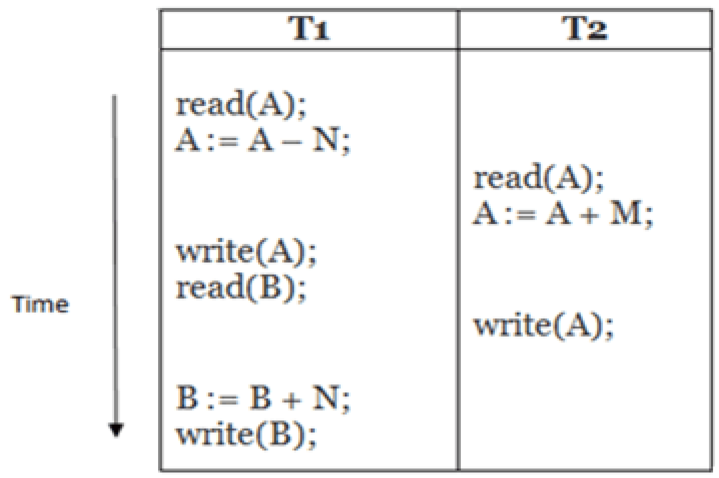
(b)



T1	T2
	read(A);
	A := A + M;
	write(A);
read(A);	
A:= A - N;	
write(A);	
read(B);	
B:= B +N;	
write(B);	

Schedule B

(c)



T1	T2

read(A);
A := A – N;	

	read(A);
	A := A + M;

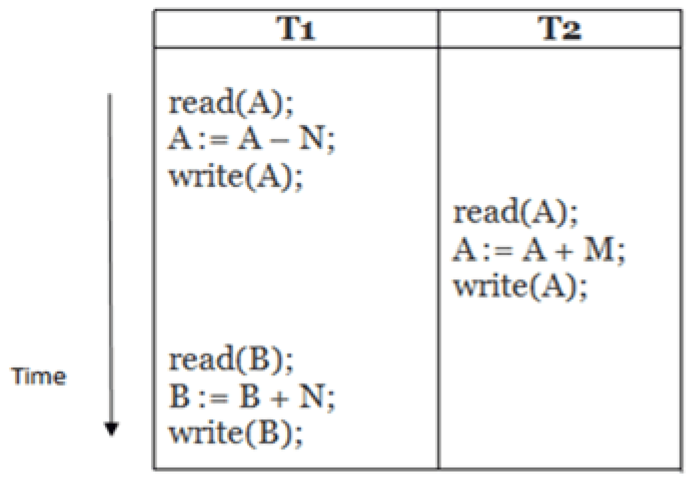
write(A);
read(B);	

	write(A);

B := B + N;
write(B);	

Schedule C

(d)



T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

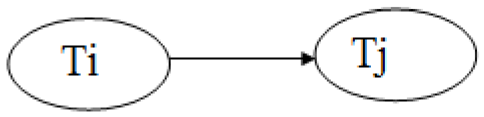
Testing of Serializability

Serialization Graph is used to test the Serializability of a schedule.

Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

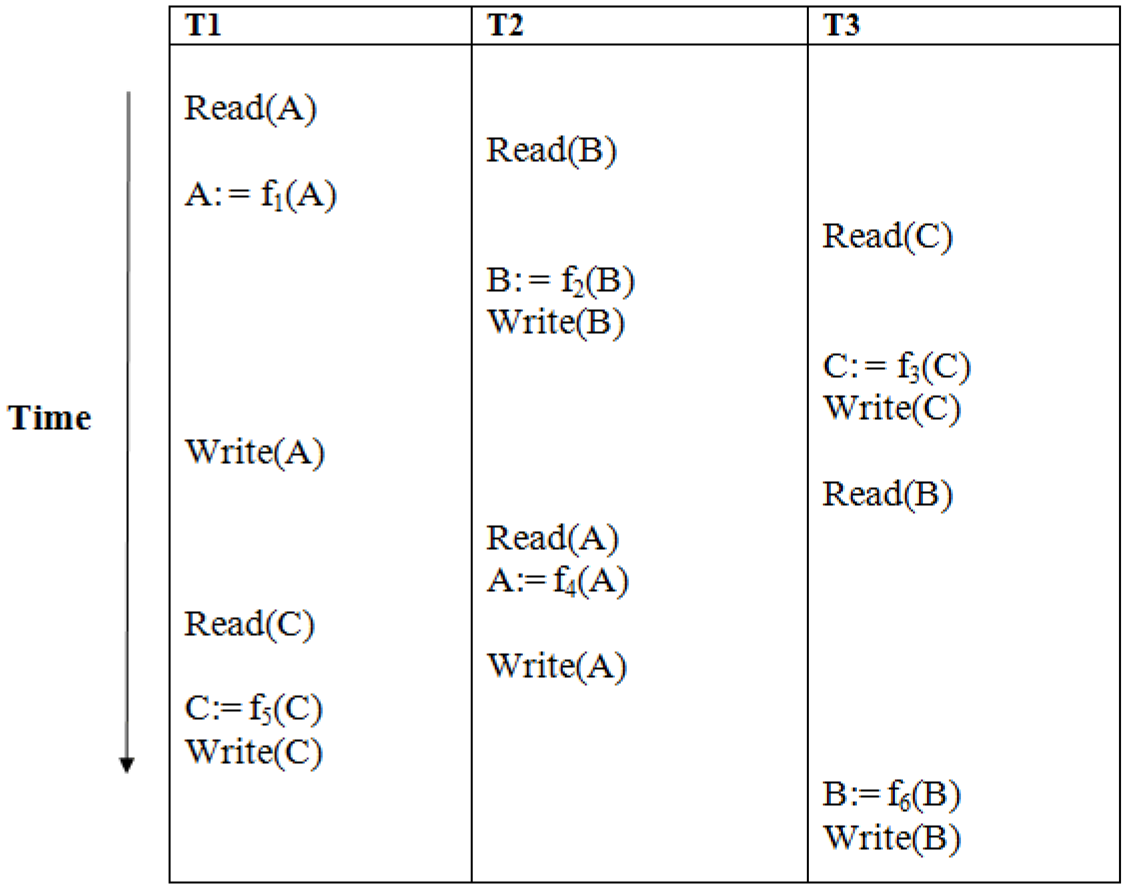
1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence graph for Schedule S



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti areexecuted before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If theprecedence graph has no cycle, then S is known as serializable.

For example:



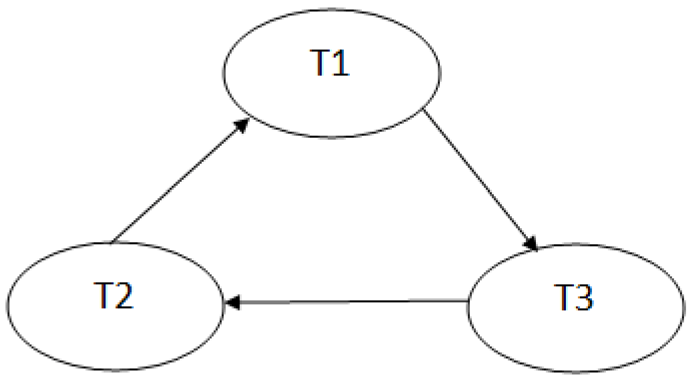
T1	T2	T3
Read(A)		
	Read(B)	
$A:=f_1(A)$		
		Read(C)
	$B:=f_2(B)$	
	Write(B)	
		$C:=f_3(C)$
		Write(C)
Write(A)		
		Read(B)
	Read(A)	
	$A:=f_4(A)$	
Read(C)		
	Write(A)	
$C:=f_5(C)$		
Write(C)		
		$B:=f_6(B)$
		Write(B)

Schedule S1

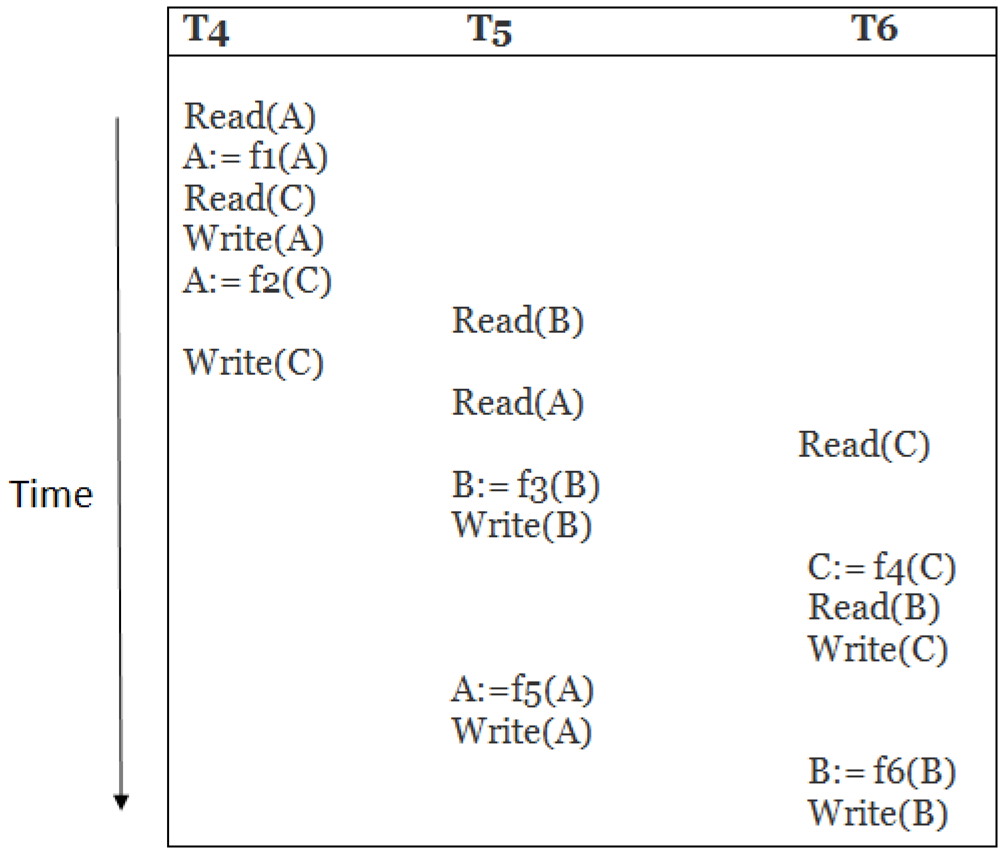
Explanation:

Read(A): In T1, no subsequent writes to A, so no new edges
Read(B): In T2, no subsequent writes to B, so no new edges
Read(C): In T3, no subsequent writes to C, so no new edges
Write(B): B is subsequently read by T3, so add edge T2 → T3
Write(C): C is subsequently read by T1, so add edge T3 → T1
Write(A): A is subsequently read by T2, so add edge T1 → T2
Write(A): In T2, no subsequent reads to A, so no new edges
Write(C): In T1, no subsequent reads to C, so no new edges
Write(B): In T3, no subsequent reads to B, so no new edges

Precedence graph for schedule S1:



The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.



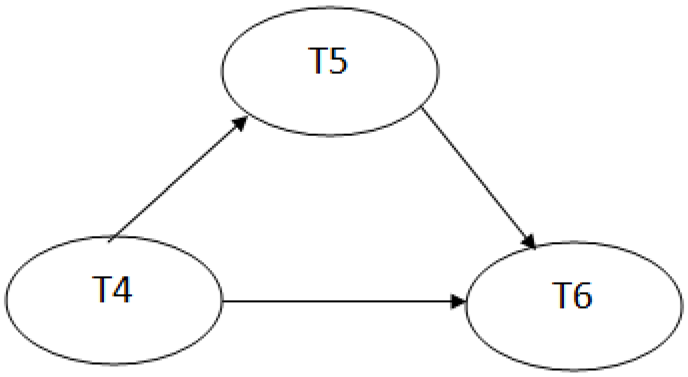
T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Schedule S2

Explanation:

Read(A): In T4,no subsequent writes to A, so no new edges
Read(C): In T4, no subsequent writes to C, so no new edges
Write(A): A is subsequently read by T5, so add edge T4 → T5
Read(B): In T5,no subsequent writes to B, so no new edges
Write(C): C is subsequently read by T6, so add edge T4 → T6
Write(B): A is subsequently read by T6, so add edge T5 → T6
Write(C): In T6, no subsequent reads to C, so no new edges
Write(A): In T5, no subsequent reads to A, so no new edges
Write(B): In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, itcan transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

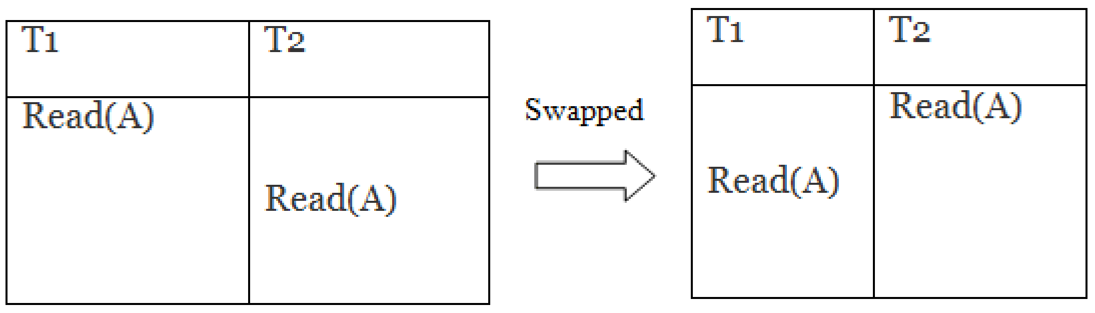
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)



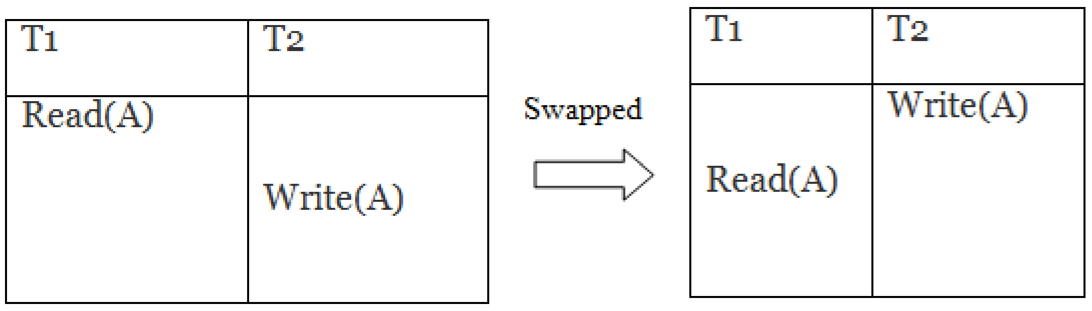
T1	T2
Read(A)	
	Read(A)

T1	T2
	Read(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



T1	T2
Read(A)	
	Write(A)

T1	T2
	Write(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 ≠ S2. That means it is conflict.

Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

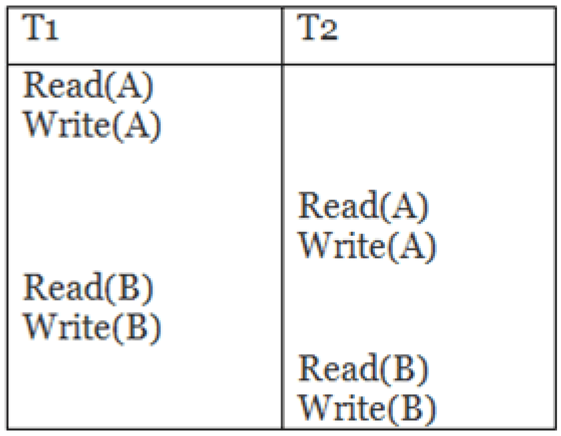
Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.

2. If each pair of conflict operations are ordered in the same way.

Example:

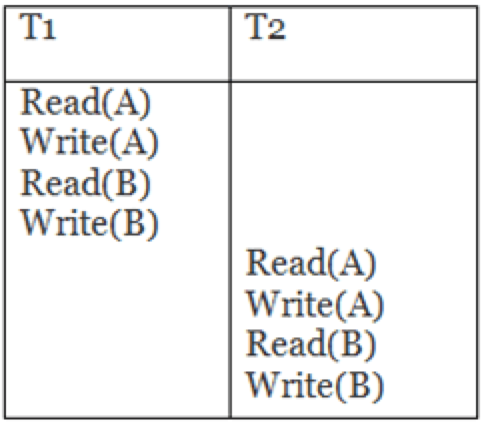
Non-serial schedule



T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule



T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Since, S1 is conflict serializable.

View Serializability

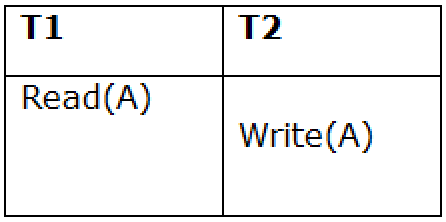
- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which d.oes not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

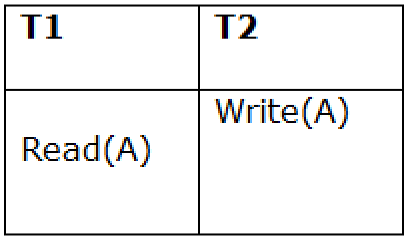
1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.



T1	T2
Read(A)	
	Write(A)

Schedule S1



T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3

Read(A)
Write(A)		

	Write(A)	
		Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback ifTj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Failure Classification

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure

2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or aninternal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The systemaborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.

  Fail-stop assumption: In the system crash, non-volatile storage is assumed notto be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was acommon problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of diskstorage.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stablestorage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied inthe database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log iswritten to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of thetransaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the databaseuntil it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database isupdated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while thetransaction is still active.
- In this technique, the database is modified immediately after every operation. It follows anactual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then theTransaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti,abort>, then the Transaction Ti needs to be undone.

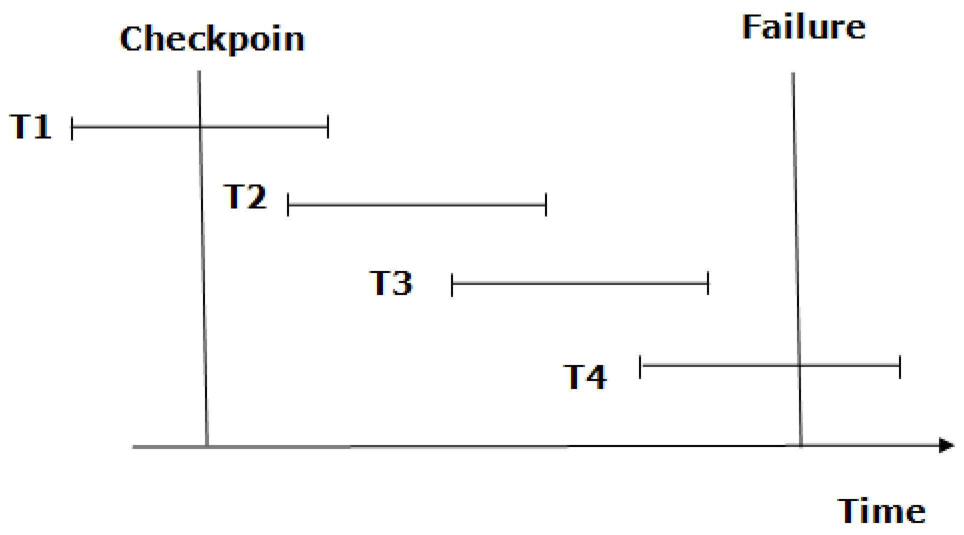
Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from thesystem and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of thetransaction, the log files will be created.

- When it reaches to the checkpoint, then the transaction will be updated into the database,and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistentstate, and all transactions were committed.

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start>and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all thetransactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why thetransaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start>but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Deadlock in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

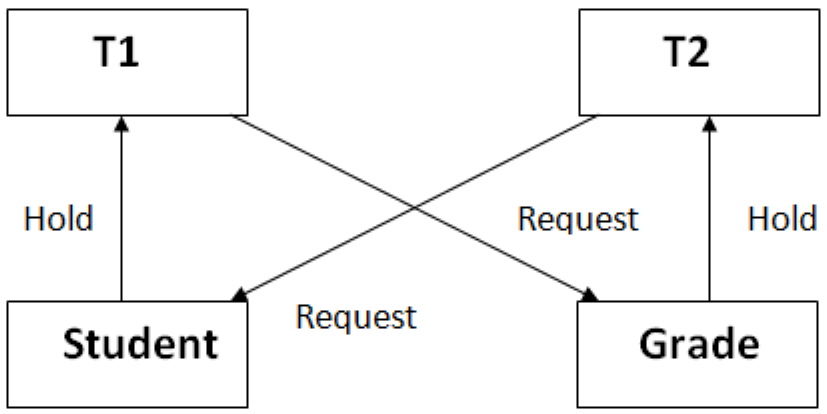


Figure: Deadlock in DBMS

Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the databaserather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this methodis suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

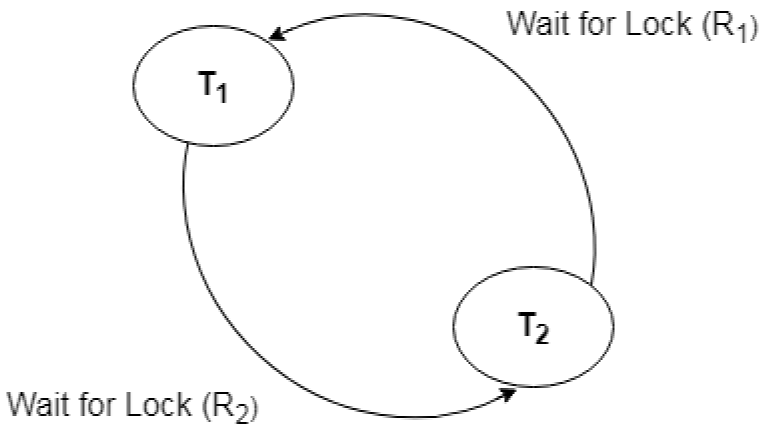
Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop,then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waitingfor some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed thattransaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. Itallows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if theolder transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj iswaiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted butwith the same timestamp.

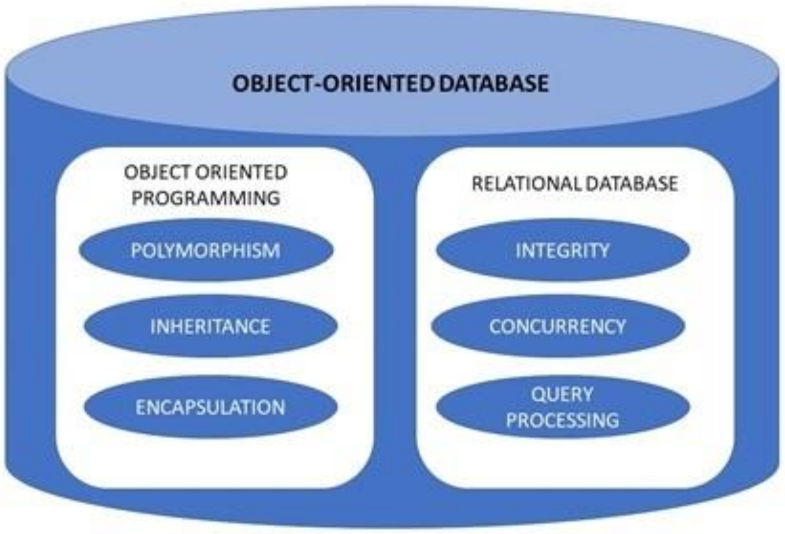
f the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until

Unit -2

Object oriented database-An object database is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.

An object-oriented database is a collection of object-oriented programming and relational database. There are various items which are created using object-oriented programming languages like C++, Java which can be stored in relational databases, but object-oriented databases are well-suited for those items.

An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.



An object-oriented database (OODBMS) or object database management system (ODBMS) is a database that is based on object-oriented programming (OOP). The data is represented and stored in the form of objects. OODBMS are also called object databases or object-oriented database management systems.

Object-Oriented Database

Object database management systems (ODBMSs) are based on objects in object-oriented programing (OOP). In OOP, an entity is represented as an object and objects are stored in memory. Objects have members such as fields, properties, and methods. Objects also have a life cycle that includes the creation of an object, use of an object, and deletion of an object. OOP has key characteristics, encapsulation, inheritance, and polymorphism. Today, there are many popular OOP languages such as C++, Java, C#, Ruby, Python, JavaScript, and Perl.

The idea of object databases was originated in 1985 and today has become common for various common OOP languages, such as C++, Java, C#, Smalltalk, and LISP. Common examples are Smalltalk is used in GemStone, LISP is used in Gbase, and COP is used in Vbase.

Object databases are commonly used in applications that require high performance, calculations, and faster results. Some of the common applications that use object databases are real-time systems, architectural & engineering for 3D modeling, telecommunications, and scientific products, molecular science, and astronomy.

Advantages of Object Databases

ODBMS provide persistent storage to objects. Imagine creating objects in your program and saving them as it is in a database and reading back from the database.

In a typical relational database, the program data is stored in rows and columns. To store and read that data and convert it into program objects in memory requires reading data, loading data into objects, and

storing it in memory. Imagine creating a class in your program and saving it as it is in a database, reading back and start using it again.
Object databases bring permanent persistent to objects. Objects can be stored in persistent storage forever.

In typical RDBMS, there is a layer of object-relational mapping that maps database schemas with objects in code. Reading and mapping an object database data to the objects is direct without any API or OR tool. Hence faster data access and better performance.

Some object database can be used in multiple languages. For example, Gemstone database supports C++, Smalltalk and Java programming languages.

Drawbacks of Object Databases

- Object databases are not as popular as RDBMS. It is difficult to find object DB developers.
- Not many programming language support object databases.
- RDBMS have SQL as a standard query language. Object databases do not have a standard.
- Object databases are difficult to learn for non-programmers.

Popular Object Databases

Here is a list of some of the popular object databases and their features.

Cache

InterSystems's Caché is a high-performance object database. Caché database engine is a set of services including data storage, concurrency management, transactions, and process management. You can think of the Caché engine as a powerful database toolkit.

Caché is also a full-featured relational database. All the data within a Caché database is available as true relational tables and can be queried and modified using standard SQL via ODBC, JDBC, or object methods. Caché is one of the fastest, most reliable, and most scalable relational databases.

Cache offers the following features,

- The ability to model data as objects (each with an automatically created and synchronized native relational representation) while eliminating both the impedance mismatch between databases and object-oriented application environments as well as reducing the complexity of relational modeling,
- A simpler, object-based concurrency model
- User-defined data types
- The ability to take advantage of methods and inheritance, including polymorphism, within the database engine
- Object-extensions for SQL to handle object identity and relationships
- The ability to intermix SQL and object-based access within a single application, using each for what they are best suited

- Control over the physical layout and clustering used to store data in order to ensure the maximum performance for applications

Cache offers a broad set of tools, which include,

- ObjectScript, the language in which most of Caché is written.
- Native implementations of SQL, MultiValue, and Basic.
- A well-developed, built-in security model
- A suite of technologies and tools that provide rapid development for database and web applications
- Native, object-based XML and web services support
- Device support (such as files, TCP/IP, printers)
- Automatic interoperability via Java, JDBC, ActiveX, .NET, C++, ODBC, XML, SOAP, Perl, Python, and more
- Support for common Internet protocols: POP3, SMTP, MIME, FTP, and so on
- A reusable user portal for your end users
- Support for analyzing unstructured data
- Support for Business Intelligence (BI)
- Built-in testing facilities

ObjectDB Object Database

ObjectDB is a powerful Object-Oriented Database Management System (ODBMS). It is compact, reliable, easy to use and extremely fast. ObjectDB provides all the standard database management services (storage and retrieval, transactions, lock management, query processing, etc.) but in a way that makes development easier and applications faster.

- ObjectDB Database Key Features
- 100% pure Java Object-Oriented Database Management System (ODBMS).
- No proprietary API - managed only by standard Java APIs (JPA 2 / JDO 2).
- Extremely fast - faster than any other JPA / JDO product.
- Suitable for database files ranging from kilobytes to terabytes.
- Supports both Client-Server mode and Embedded mode.
- Single JAR with no external dependencies.
- The database is stored as a single file.
- Advanced querying and indexing capabilities.
- Effective in heavy loaded multi-user environments.
- Can easily be embedded in applications of any type and size.
- Tested with Tomcat, Jetty, GlassFish, JBoss, and Spring.

ObjectDatabase++

ObjectDatabase++ (ODBPP) is an embeddable object-oriented database designed for server applications that require minimal external maintenance. It is written in C++ as a real-time ISAM level database with the ability to auto recover from system crashes while maintaining database integrity.

Objectivity/DB

Objectivity/DB is a scalable, high performance, distributed Object Database (ODBMS). It is extremely good at handling complex data, where there are many types of connections between objects and many variants.

Objectivity/DB runs on 32 or 64-bit processors running Linux, Mac OS X, UNIX (Oracle Solaris) or Windows.

There are C++, C#, Java and Python APIs.

All platform and language combinations are interoperable. For example, objects stored by a program using C++ on Linux can be read by a C# program on Windows and a Java program on Mac OS X.

Objectivity/DB generally runs on POSIX file systems, but there are plugins that can be modified for other storage infrastructure.

Objectivity/DB client programs can be configured to run on a standalone laptop, networked workgroups, large clusters or in grids or clouds with no changes to the application code.

Versant Object Database

Versant Object-Oriented Database is an object database that supports native object persistence and used to build complex and high-performance data management systems.

Key Benefits

- Real-time analytical performance
- Big Data management
- Cut development time by up to 40%
- Significantly lower total ownership cost
- High availability

Object-relational Databases

Object-relational database (ORD), or object-relational database management systems (ORDBMS) are databases that support both objects and relational database features. OR databases are relational database management systems with the support of an object-oriented database model. That means, the entities are represented as objects and classes and OOP features such as inheritance are supported in database schemas and in the query language.

PostgreSQL is the most popular pure ORDBMS. Some popular databases including Microsoft SQL Server, Oracle, and IBM DB2 also support objects and can be considered as ORDBMS.

<u>distributed database</u>- *A distributed database is a database in which storage devices are not all attached to a common processor. It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers.(source wiki)* One of the strongest fault tolerance techniques of a distributed database is data replication, which is a process of storing separate copies of the database or tables at two or more sites.

Understanding Vertical and Horizontal fragmentation
Vertical fragmentation

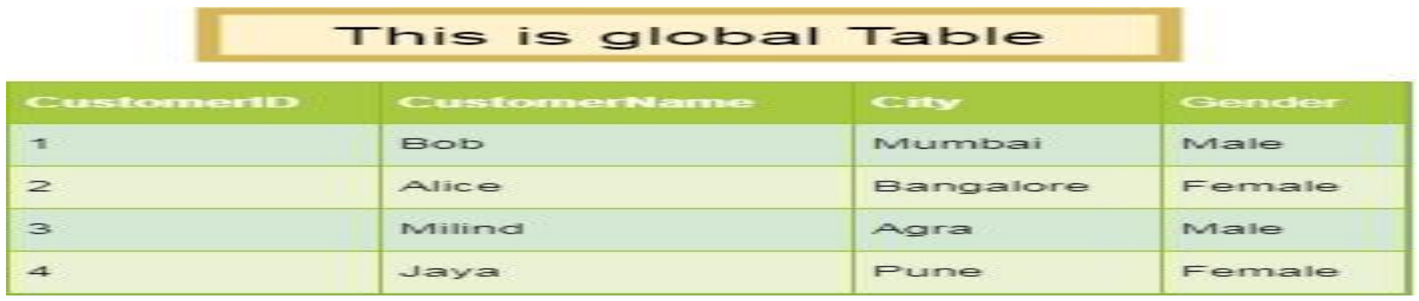
1. Vertical fragmentation is a subset of attributes.
2. Basically, vertical fragmentation splits tables by columns

Horizontal Fragmentation

1. Horizontal Fragmentation is a subset of tuples (rows).
2. Horizontal Fragmentation splits tables by rows.

Example

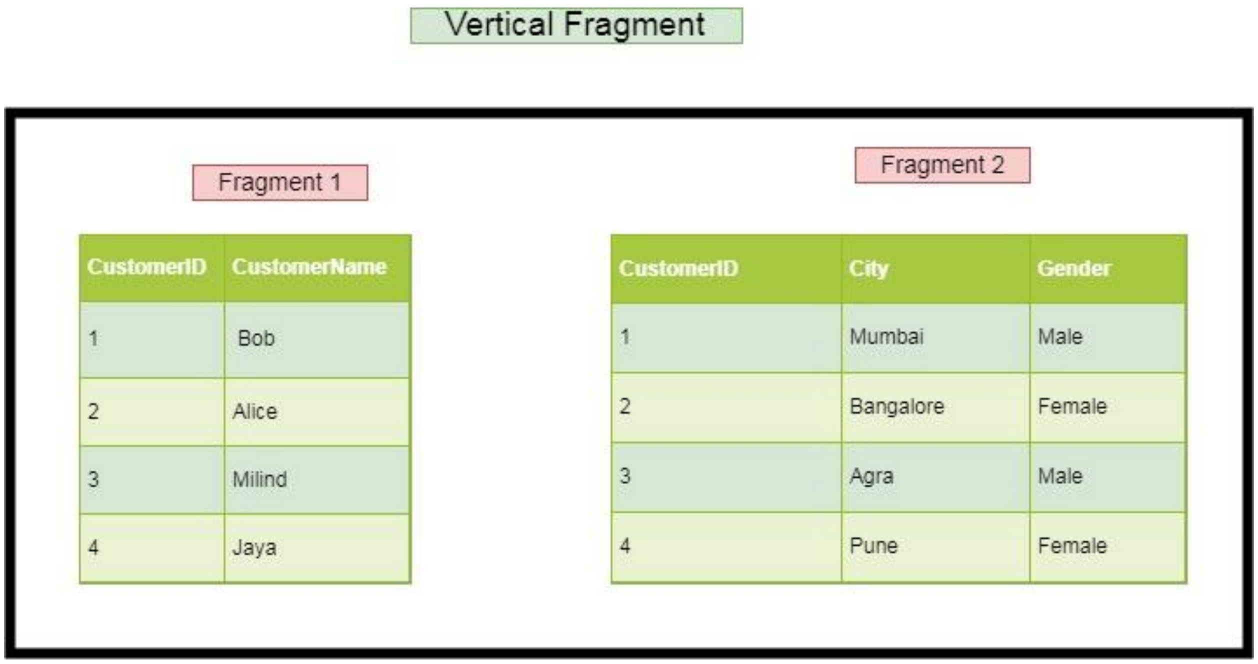
Let's say I have one global table (e.g. Customer Table):



This is global Table

CustomerID	CustomerName	City	Gender
1	Bob	Mumbai	Male
2	Alice	Bangalore	Female
3	Milind	Agra	Male
4	Jaya	Pune	Female

Vertical fragmentation would be like this: (Here, we are storing 2 columns at one fragment and 3 columns at another fragment, however, id is important at both sites because it's a primary key)



Vertical Fragment

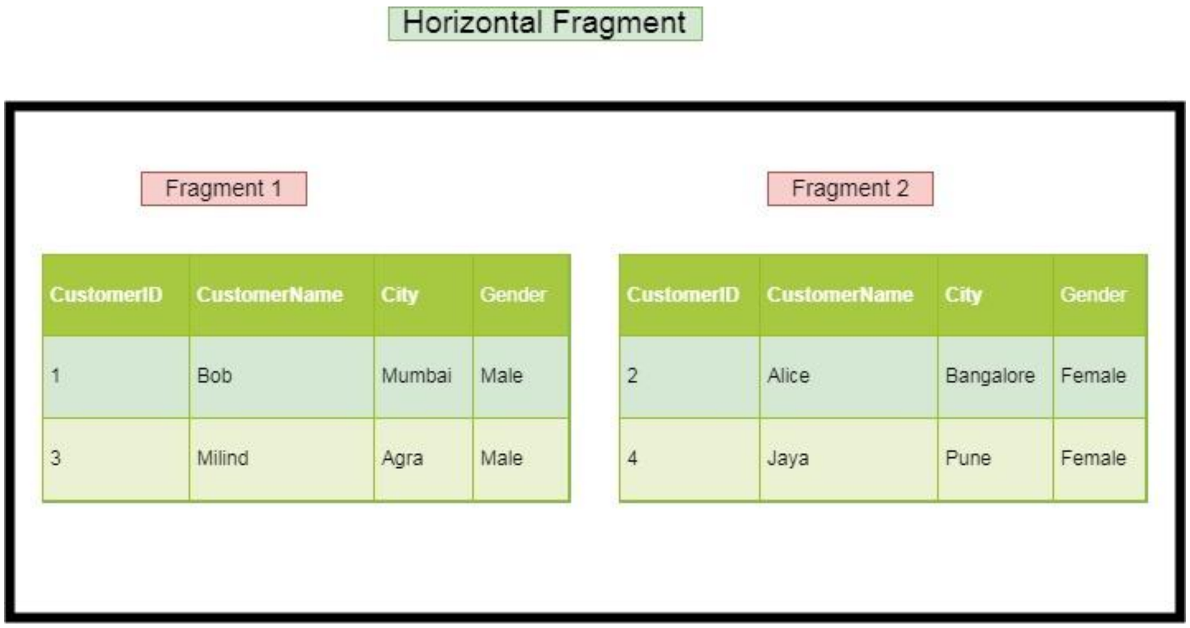
Fragment 1

CustomerID	CustomerName
1	Bob
2	Alice
3	Milind
4	Jaya

Fragment 2

CustomerID	City	Gender
1	Mumbai	Male
2	Bangalore	Female
3	Agra	Male
4	Pune	Female

Horizontal fragment looks like this: (Here, we are diving fragment based on some condition such that all data with gender male will reside at one fragment and others at different fragment).



Horizontal Fragment

Fragment 1

CustomerID	CustomerName	City	Gender
1	Bob	Mumbai	Male
3	Milind	Agra	Male

Fragment 2

CustomerID	CustomerName	City	Gender
2	Alice	Bangalore	Female
4	Jaya	Pune	Female

Understanding with an example

I am using SQL Plus to perform these operations. I have two machines m1 and m2; on m1 I am creating global table whereas on m2 I am storing fragment values.

Vertical Fragmentaion

There is one global table and the secondary table is given as,

This is for tblCust,

Global

```
create table tblCust_glo
(
   Cid varchar2(10) primary key,
   Cname varchar2(10) not null,
   Ctype varchar2(10) not null,
   Cmob integer not null
);
```

Secondary

```
create table tblCust_1
(
   Cid varchar2(10) primary key,
   Cname varchar2(10) not null,
   Ctype varchar2(10) not null
);

```

Creating link from one node to another

CREATE DATABASE LINK

```
Create database link linker
connect to scott identified by
tiger using ‘IT_78’;
```

--Here IT_78 is net service name of my machine.
--You can create it by visiting this link.

Trigger

```
create or replace trigger trigCust_glo
after insert on tblCust_glo
for each row
begin
insert into tblCust_1@linker
   values(:new.Cid,:new.Cname,:new.Ctype);
end;
```

```
/
```

This is for tblVehical.

Global

```
create table tblVehicle_glo
(
   Vid varchar2(10) primary key,
   Vclass varchar2(10) not null,
   Vrgis varchar2(10) not null,
   Vodo integer not null,
   Vmeter integer not null,
   Vstatus varchar2(10) not null
);
```

Secondary

```
create table tblVehicle_1
(
   Vid varchar2(10) primary key,
   Vclass varchar2(10) not null,
   Vmeter integer not null,
   Vstatus varchar2(10) not null
);


--Trigger:-
create or replace trigger trigVeh_glo
after insert on tblVehicle_glo
for each row
begin
insert into tblVehicle_1@linker
   values(:new.Vid,:new.Vclass,:new.Vmeter,:new.Vstatus);
end;
/
```

Horizontal Fragmentation

```
Create table customer_horizontal as
(select * from tblCust_glo@linker Where Ctype='Premium')
```

Client-server database system is a system in which server manages the resources and client consumed these resources.

Functionality provided by database systems can be broadly divided into two parts - the front end and the back end. The back end manages access structures, query evaluation , optimization , concurrency control and recovery. The front end of a database consists of tools such as SQL user interface . The interface between the front end and back end is through SQL or application program.

There are two types of client server database system are given below :

1. Two tier client-server database system
2. Three tier client-server database system

> ***Client/Server architecture*** of database system has two logical components namely client, and server. Clients are generally personal computers or workstations whereas server is large workstations, mini range computer system or a mainframe computer system. The applications and tools of DBMS run on one or more client platforms, while the DBMS software reside on the server. The server computer is called back end and the client's computer is called front end.
> These server and client computers are connected into a network. The applications and tools act as clients of the DBMS, making requests for its services. The DBMS, in turn, processes these requests and returns the results to the client(s). Client/Server architecture handles the Graphical User Interface (GUI) and does computations and other programming of interest to the end user. The server handles parts of the job that are common to many clients, for example, database access and updates.

Data Dictionary

A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

In a relational database, the metadata in the data dictionary includes the following:

- Names of all tables in the database and their owners
- Names of all indexes and the columns to which the tables in those indexes relate
- Constraints defined on tables, including primary keys, foreign-key relationships to other tables, and not-null constraints

For most relational database management systems (RDBMS), the database management system software needs the data dictionary to access the data within a database. For example, the Oracle DB software has to read and write to an Oracle DB. However, it can only do this via the data dictionary created for that particular database.

For instance, suppose that in a commercial bank's database, the administrator wants to determine which table holds information about loans. Making an educated guess that the table most likely has the word "LOAN" in it, he would issue the following query on the data dictionary (the first query is for an Oracle DB, while the second is for an SQL Server DB):

- SELECT * FROM DBA_TABLES WHERE TABLE_NAME LIKE '%LOAN%';
- SELECT * FROM SYSOBJECTS WHERE TYPE='U' AND NAME LIKE '%LOAN%';

Database Objects

Two small but important distinctions in database objects are needed:

- An object type is the base concept or idea of an object; for example, the concept of a table or index.
- An object instance is an example of an object type. For example, a table called CUSTOMER_MASTER is an instance of the object type TABLE.

Most of the major database engines offer the same set of major database object types:

- Tables
- Indexes
- Sequences
- Views
- Synonyms

Although there are subtle variations in the behavior and the syntax used for the creation of these major database object types, they are almost identical in their concept and what they mean. A table in Oracle behaves almost exactly as a table in SQL Server. This makes work much easier for the database administrator. It is analogous to moving from one car to another made by a different manufacturer; the switches for turning the headlights on may be in different locations, but the overall layout is broadly similar.

When creating an object instance, it is a good idea to follow an easy-to-understand naming convention. This is especially important for database designers whose products are intended to be used by several people. It is also helpful to make work as simple as possible for in-house database administrators by reducing the number of queries made to the creator later. A simple guideline is to add suffixes. Here are two examples:

- Suffix all the master tables using _MASTER:
  - CUSTOMER_MASTER
  - ACCOUNTS_MASTER
  - LOANS_MASTER
- Suffix all transactional tables using the suffix _TRANS:
  - DAILY_TRANS
  - LOANS_TRANS
  - INTERBANK_TRANS

Data Modeling

Data modeling is a representation of the data structures in a table for a company’s database and is a very powerful expression of the company's business requirements. This data model is the guide used by functional and technical analysts in the design and implementation of a database.

Data models are used for many purposes, from high-level conceptual models to physical data models

Data Modeling

Data modeling explores data-oriented structures and identifies entity types. This is unlike class modeling, where classes are identified.

Three basic styles of data modeling are generally used in practice today.

- Conceptual Data Models: High-level, static business structures and concepts
- Logical Data Models (LDMs): Entity types, data attributes and relationships between entities
- Physical Data Models (PDMs): The internal schema database design

Database Model

A database model refers to the logical structure, representation or layout of a database and how the data will be stored, managed and processed within it. It helps in designing a database and serves as blueprint for application developers and database administrators in creating a database.

Database Model

A database model is primarily a type of data model. Depending on the model in use, a database model can include entities, their relationships, data flow, tables and more. For example, within a hierarchal database mode, the data model organizes data in the form of a tree-like structure having parent and child segments.

Some of the popular database models include relational models, hierarchical models, flat file models, object oriented models, entity relationship models and network models.

Definition - *Relational Data Model*

A relational data model involves the use of data tables that collect groups of elements into relations. These models work based on the idea that each table setup will include a primary key or identifier. Other tables use that identifier to provide "relational" data links and results. Database administrators use something called Structured Query Language (SQL) to retrieve data elements from a relational database.

Other aspects of relational database design correspond to specific parts of a data table. For example, a conventional database row would represent a tuple, which is a set of data that revolves around a particular instance or virtual object so that the primary key is its unique identifier. A column name in a data table is associated with an attribute, an identifier or feature that all parts of a data set have. These and other strict conventions help to provide database administrators and designers with standards for crafting relational database setups.

As mentioned, the primary key is a fundamental tool in creating and using relational data models. It must be unique for each member of a data set. It must be populated for all members. Inconsistencies can cause problems in how developers retrieve data. Other issues with relational database designs include excessive duplication of data, faulty or partial data, or improper links or associations between tables. A large part of routine database administration involves evaluating all of the data sets in a database to make sure that they are consistently populated and will respond well to SQL or any other data retrieval method.

Hierarchical Database

The idea behind hierarchical database models is useful for a certain type of data storage, but it is not extremely versatile. Its limitations mean that it is confined to some very specific uses. For example, where each individual person in a company may report to a given department, the department can be used as a parent record and the individual employees will represent secondary records, each of which links back to that one parent record in a hierarchical structure.

Hierarchical databases were popular in early database design, in the era of mainframe computers. While some IBM and Microsoft models are still in use, many other types of business databases use more flexible models to accommodate more sophisticated types of data management. Hierarchical models make the most sense where the primary focus of information gathering is on a concrete hierarchy such as a list of business departments, assets or people that will all be associated with specific higher-level primary data elements.

Definition - *Network Model*

A network model is a database model that is designed as a flexible approach to representing objects and their relationships. A unique feature of the network model is its schema, which is viewed as a graph where relationship types are arcs and object types are nodes. Unlike other database models, the network model's schema is not confined to be a lattice or hierarchy; the hierarchical tree is replaced by a graph, which allows for more basic connections with the nodes.

Charles Bachman was the original inventor of the network model. In 1969, the Conference on Data Systems Languages (CODASYL) Consortium developed the network model into a standard specification. A second publication was introduced in 1971, which later turned into the basis for virtually all implementations.

The benefits of the network model include:

- Simple Concept: Similar to the hierarchical model, this model is simple and the implementation is effortless.
- Ability to Manage More Relationship Types: The network model has the ability to manage one-to-one (1:1) as well as many-to-many (N: N) relationships.
- Easy Access to Data: Accessing the data is simpler when compared to the hierarchical model.
- Data Integrity: In a network model, there's always a connection between the parent and the child segments because it depends on the parent-child relationship.
- Data Independence: Data independence is better in network models as opposed to the hierarchical models.

The drawbacks of the network model include:

- System Complexity: Each and every record has to be maintained with the help of pointers, which makes the database structure more complex.
- Functional Flaws: Because a great number of pointers is essential, insertion, updates, and deletion become more complex.

- Lack of Structural Independence: A change in structure demands a change in the application as well, which leads to lack of structural independence.

Definition - *Normalization*

Normalization is the process of reorganizing data in a database so that it meets two basic requirements: (1) There is no redundancy of data (all data is stored in only one place), and (2) data dependencies are logical (all related data items are stored together). Normalization is important for many reasons, but chiefly because it allows databases to take up as little disk space as possible, resulting in increased performance.

Normalization is also known as data normalization.

Normalization

The three main types of normalization are listed below. Note: "NF" refers to "normal form."

- 1NF
- 2NF
- 3NF

The following three NFs exist but are rarely used:

- BCNF
- 4NF
- 5NF

The first three NFs were derived in the early 1970s by the father of the relational data model, E.F. Codd. Almost all of today's relational database engines use his rules.

Definition - *First Normal Form (1NF)*

First normal form (1NF) sets the fundamental rules for database normalization and relates to a single table within a relational database system. Normalization follows three basic steps, each building on the last. The first of these is the first normal form.

The first normal form states that:

- Every column in the table must be unique
- Separate tables must be created for each set of related data
- Each table must be identified with a unique column or concatenated columns called the primary key
- No rows may be duplicated
- no columns may be duplicated
- no row/column intersections contain a null value
- no row/column intersections contain multivalued fields

Explanation

First Normal Form (1NF)

The first step in confirming 1NF is modifying multivalued columns to make sure that each column in a table does not take more than one entry.

Searching records with duplicate entries is complex. To overcome this situation, all records involved in a relational database table have to be identified by a unique value which will have a separate column (or attribute). This unique key is called an index key and is used to locate data for retrieval or other manipulation.
Having a unique key does not resolve the requirements of 1NF. According to the rules, there can be no multiple entries into a single field. For example, in a data table of customer information, a single field could be allowed to store multiple entries, such as where a customer has multiple telephone numbers. This is a violation of the 1NF rules. This particular problem in our example can be resolved by creating a customer ID index in the main table and then adding a separate table that has a column for the telephone numbers and another column for the customer ID.
This allows proper use of relational queries to extract data from a relational database. Null, or multiple entry fields both cause issues with data manipulation and extraction so the normalizing process removes ambiguity.
Removing repeating values from a table is the next step toward first normalized form. Repeating values can be moved to a new table.
The final step in implementing first normal form is maintaining atomicity of data. Each individual field should hold the smallest data element possible to facilitate easy sorting and searching. For instance, the date column can be separated into day, month and year.
Tables satisfying first normal form can also contain functionally dependent fields. Functional dependency exists between two fields when the value in field 1 determines the value in field 2 and there is only one value in field 2. In such a case, field 2 is functionally dependent on field 1.
Tables satisfying the higher normal forms (second, third and fourth) necessarily follow first normal form but the reverse is not true. All tables complying with first normal form may not follow the higher normal forms, as the higher normal forms include even more stringent rules.

Definition - *Second Normal Form (2NF)*

Second normal form (2NF) is the second step in normalizing a database. 2NF builds on the first normal form (1NF).

Normalization is the process of organizing data in a database so that it meets two basic requirements:

- There is no redundancy of data (all data is stored in only one place).
- Data dependencies are logical (all related data items are stored together).

A 1NF table is in 2NF form if and only if all of its non-prime attributes are functionally dependent on the whole of every candidate key.

Explains *Second Normal Form (2NF)*

After meeting the requirements of 1NF, 2NF requires the database designer to do the following:

1. Split up all data resulting in many-to-many relationships and store the data as separate tables. For example, in a database used by a school's application, two of the tables are STUDENT and SUBJECT. In real life, a student takes several subjects simultaneously while a subject is studied by several students. These are many-to-many relationships. 2NF states that this relationship must be split into more than the two tables above (STUDENT and SUBJECT). One way of splitting them is by introducing a third table, which contains the columns Student ID, Subject ID, Semester and Year. In this way, there is no direct relationship between STUDENT and SUBJECT because all relationships are created indirectly through the third table.
2. Create relationships between tables by use of foreign keys. For example, a bank's database contains two tables: CUSTOMER_MASTER (for storing customer details) and ACCOUNT_MASTER (for storing details about bank accounts, including which customer holds which account). There must be a way to link the two tables to know who the customer is for each account. The way to do this is via a foreign key, which is a column in the ACCOUNT_MASTER table pointing to a corresponding column in the CUSTOMER_MASTER table.

A table for which there are no partial functional dependencies on the primary key might or might not be in 2NF. In addition to the primary key, the table may contain other candidate keys; it is necessary to establish that no non-prime attributes have part-key dependencies on any of these candidate keys.

Definition - *Third Normal Form (3NF)*

Third normal form (3NF) is the third step in normalizing a database and it builds on the first and second normal forms, 1NF and 2NF.

3NF states that all column reference in referenced data that are not dependent on the primary key should be removed. Another way of putting this is that only foreign key columns should be used to reference another table, and no other columns from the parent table should exist in the referenced table.

Explains *Third Normal Form (3NF)*

Consider a bank's database, which contains two tables: CUSTOMER_MASTER for storing customer details and ACCOUNT_MASTER for storing details about bank accounts, including which customer holds which account. In this case, there needs to be a way to link the two tables in order to tie an account to the customer who owns it. The way to do this is via a foreign key. This is a column in the ACCOUNT_MASTER table that points to or references a corresponding column (called the primary key) in the CUSTOMER_MASTER parent table. Let's call this column CustID.

Suppose that customer Andrew Smith creates an account in the CUSTOMER_MASTER table with CustID 20454. Mr. Smith holds a savings account with the number S-200802-005, whose details are stored in the ACCOUNT_MASTER table. This means that the ACCOUNT_MASTER table will have a column called CustID, which is not an original piece of data. Instead, it also has the value 20454, which simply references the same CustID in the CUSTOMER_MASTER table.

Now, 3NF dictates that in our ACCOUNT_MASTER table, the only information we hold about the customer should be the CustID (20454) as a foreign key, and it refers to and identifies the customer who owns this same CustID in the CUSTOMER_MASTER table (Andrew Smith). No other data about our customer (such as name, date of birth, gender and so on) should be stored in the ACCOUNT_MASTER table, or indeed any other table, because all this data about him is already stored in CUSTOMER_MASTER. By doing this, the only customer data stored outside the CUSTOMER_MASTER table is the CustID. This pays handsome dividends by ensuring there is no data duplication, which in turn makes queries run much more efficiently and reduces the amount of storage required.

Definition - *Fourth Normal Form (4NF)*

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key.

It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF; it must not contain more than one multivalued dependency.

Explains *Fourth Normal Form (4NF)*

A multivalued dependency is best illustrated using an example. In a table containing a list of three things - college courses, the lecturer in charge of each course and the recommended book for each course - these three elements (course, lecturer and book) are independent of one another. Changing the course's recommended book, for instance, has no effect on the course itself. This is an example of multivalued dependency: An item depends on more than one value. In this example, the course depends on both lecturer and book.

Definition - *Boyce-Codd Normal Form (BCNF)*

Boyce-Codd Normal Form (BCNF) is one of the forms of database normalization. A database table is in BCNF if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key.

BCNF is also sometimes referred to as 3.5NF, or 3.5 Normal Form.

Explains *Boyce-Codd Normal Form (BCNF)*

BCNF was developed by Raymond Boyce and E.F. Codd; the latter is widely considered the father of relational database design.

BCNF is really an extension of 3rd Normal Form (3NF). For this reason it is frequently termed 3.5NF. 3NF states that all data in a table must depend only on that table's primary key, and not on any other field in the table. At first glance it would seem that BCNF and 3NF are the same thing. However, in some rare cases it does happen that a 3NF table is not BCNF-compliant. This may happen in tables with two or more overlapping composite candidate keys.

Unit -3

Database administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software often hire specialized information technology personnel called database administrators or DBAs.

DDL- Data Definition Language (DDL) statements are used to define the database structure or schema. Data Definition Language understanding with database schemas and describes how the data should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE belongs to the DDL. DDL is about "metadata".

DDL includes commands such as CREATE, ALTER and DROP statements.DDL is used to CREATE, ALTER OR DROP the database objects (Table, Views, Users).

Data Definition Language (DDL) are used different statements:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

CREATE TABLE
Syntax: Create table table name(fieldname1 datatype(),fieldname2 datatype()...);

ALTER TABLE
1. ADD
2.MODIFY

ADD
Syntax:alter table table name ADD (fieldname datatype()...);

modify
syntax: Alter table table name modify (fieldname datatype()...);

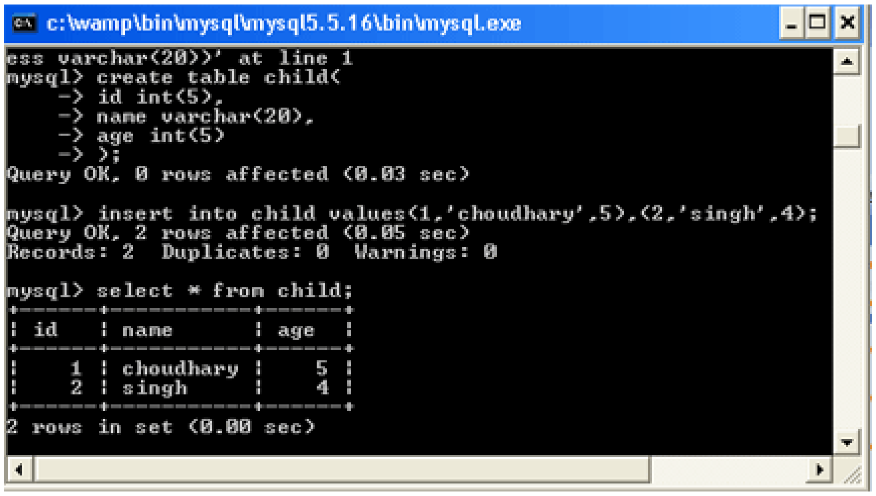
DESCRIBE TABLE
Syntax: DESCRIBE TABLE NAME;

DROP TABLE
Syntax: DROP Table name;

COMMENT - add comments to the data dictionary

RENAME - rename a table
Synatax: rename table table name to new table name

Examples : In this example we creates a table and insert the values.

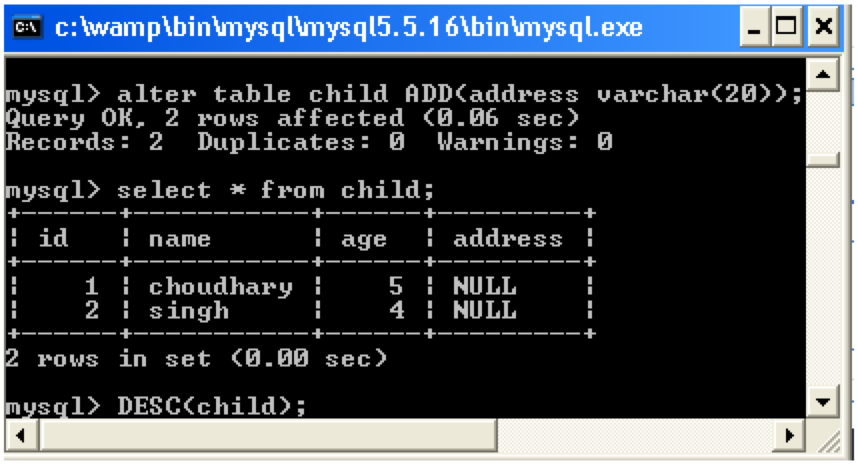


```
c:\wamp\bin\mysql\mysql5.5.16\bin\mysql.exe
ess varchar(20))' at line 1
mysql> create table child(
    -> id int(5),
    -> name varchar(20),
    -> age int(5)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> insert into child values(1,'choudhary',5),(2,'singh',4);
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from child;
+------+-----------+------+
| id   | name      | age  |
+------+-----------+------+
|    1 | choudhary |    5 |
|    2 | singh     |    4 |
+------+-----------+------+
2 rows in set (0.00 sec)
```

Example : In the following figure shows the alter a table .



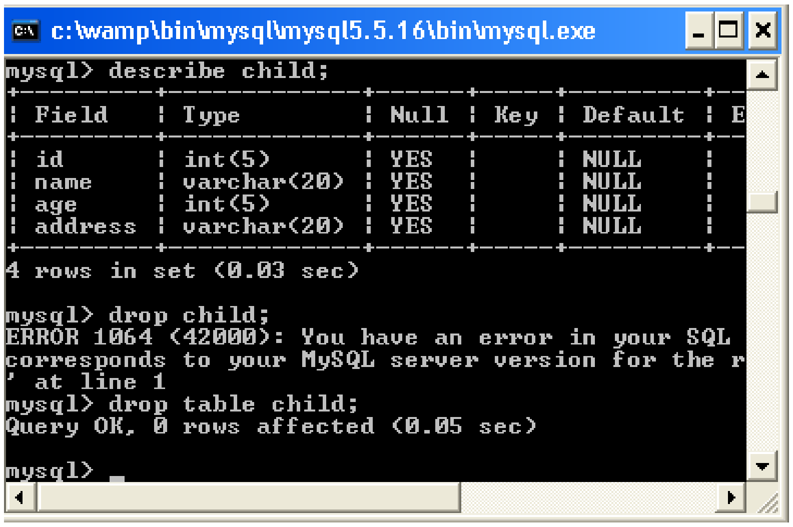
```
c:\wamp\bin\mysql\mysql5.5.16\bin\mysql.exe

mysql> alter table child ADD(address varchar(20));
Query OK, 2 rows affected (0.06 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from child;
+------+-----------+------+---------+
| id   | name      | age  | address |
+------+-----------+------+---------+
|    1 | choudhary |    5 | NULL    |
|    2 | singh     |    4 | NULL    |
+------+-----------+------+---------+
2 rows in set (0.00 sec)

mysql> DESC(child);
```

For Example : In this figure shows the describe command.



```
c:\wamp\bin\mysql\mysql5.5.16\bin\mysql.exe
mysql> describe child;
+---------+-------------+------+-----+---------+--
| Field   | Type        | Null | Key | Default | E
+---------+-------------+------+-----+---------+--
| id      | int(5)      | YES  |     | NULL    |
| name    | varchar(20) | YES  |     | NULL    |
| age     | int(5)      | YES  |     | NULL    |
| address | varchar(20) | YES  |     | NULL    |
+---------+-------------+------+-----+---------+--
4 rows in set (0.03 sec)

mysql> drop child;
ERROR 1064 (42000): You have an error in your SQL
corresponds to your MySQL server version for the r
' at line 1
mysql> drop table child;
Query OK, 0 rows affected (0.05 sec)

mysql> _
```

DML- Data Manipulation Language (DML) statements are used for managing data within schema objects DML deals with data manipulation, and therefore includes most common SQL statements such SELECT, INSERT, etc. DML allows to add / modify / delete data itself.
DML is used to manipulate with the existing data in the database objects (insert, select, update, delete).

DML Commands:
1.INSERT
2.SELECT
3.UPDATE
4.DELETE

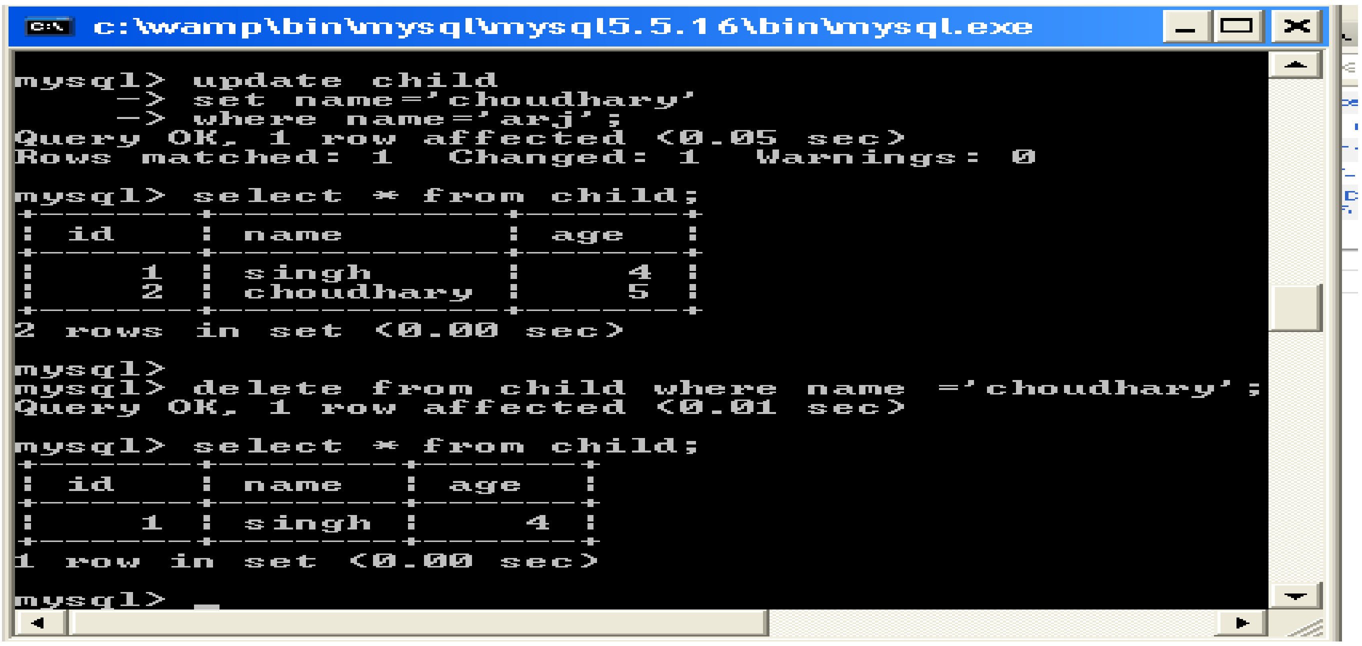
***INSERT:**
Syntax: INSERT INTO Table name values();

***SELECT:**
Syntax: Select*from <table name>

***UPDATE:**
Syntax: Update<table name> set to(calculation);

***DELETE:**
Syntax: Delete form<table name>

Example : In the below figure shows the update and delete command on the given table.



```
mysql> update child
    -> set name='choudhary'
    -> where name='arj';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from child;
+------+-----------+------+
| id   | name      | age  |
+------+-----------+------+
|    1 | singh     |    4 |
|    2 | choudhary |    5 |
+------+-----------+------+
2 rows in set (0.00 sec)

mysql>
mysql> delete from child where name ='choudhary';
Query OK, 1 row affected (0.01 sec)

mysql> select * from child;
+------+-------+------+
| id   | name  | age  |
+------+-------+------+
|    1 | singh |    4 |
+------+-------+------+
1 row in set (0.00 sec)

mysql> _
```

DCL- DCL is the abstract of Data Control Language. Data Control Language includes commands such as GRANT, and concerns with rights, permissions and other controls of the database system. DCL is used to grant / revoke permissions on databases and their contents. DCL is simple, but MySQL permissions are a bit complex.

DCL is about security. DCL is used to control the database transaction.DCL statement allow you to control who has access to specific object in your database.

1. GRANT
2. REVOKE

GRANT : It provides the user's access privileges to the database. In the MySQL database offers both the administrator and user a great extent of the control options. By the administration side of the process includes the possibility for the administrators to control certain user privileges over the MySQL server by restricting their access to an entire the database or ust limiting permissions for a specific table.It Creates an entry in the security system that
allows a user in the current database to work with data in the current database or execute specific statements.

Syntax :
Statement permissions:

GRANT { ALL | statement [,...n] }
TO security_account [,...n]

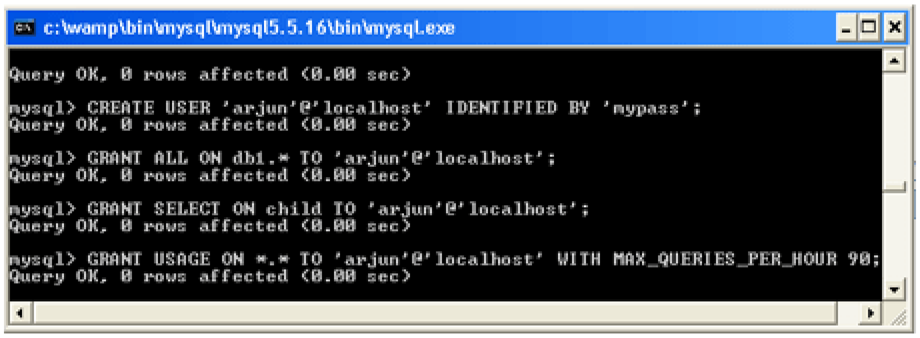
Normally, a database administrator first uses CREATE USER to create an account, then GRANT to define its privileges and characteristics.

For example:
CREATE USER 'arjun'@'localhost' IDENTIFIED BY 'mypass';

GRANT ALL ON db1.* TO 'arjun'@'localhost';

GRANT SELECT ON child TO 'arjun'@'localhost';

GRANT USAGE ON *.* TO 'arjun'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;



```
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE USER 'arjun'@'localhost' IDENTIFIED BY 'mypass';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL ON db1.* TO 'arjun'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT ON child TO 'arjun'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT USAGE ON *.* TO 'arjun'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
Query OK, 0 rows affected (0.00 sec)
```

REVOKE : The REVOKE statement enables system administrators and to revoke the privileges from MySQL accounts.

Syntax : REVOKE
priv_type [(column_list)]
[, priv_type [(column_list)]] ...
ON [object_type] priv_level
FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION

```
FROM user [, user] ...
```

For example:

```
mysql> REVOKE INSERT ON *.* FROM 'arjun'@'localhost';
```

Database Manager

The data manager is the central software component of the DBMS. It is sometimes referred to as the database control system. One of the functions of the data manager is to convert operations in the user's queries coming directly via the query processor or\ indirectly via an application program from the user's logical view to a physical file system. The data manager is responsible for interfacing with the file system as show. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data, as well as its security, are also performed by the data manager. It is also the responsibility of the Data. Manager to provide the synchronization in the simultaneous operations performed by concurrent users and to maintain the backup and recovery operations.

Database Manager Responsibility

The role comes with responsibility and you'll be expected to oversee projects and ensure all members of your team are working on the right thing at the right time to keep databases in order. Often, you'll be expected to determine the best possible method of organizing data, recording, then implementing it. The type of database and data you'll deal with will depend on your employer or client. For example, you could be helping a gym maintain records of its members for example, or develop the most effective way for a commercial client to deal with its invoices. Because so many businesses and sectors rely on technology, there is the opportunity to work alongside a wide range of organizations. This means as a database manager your work can involve a lot of variety and give you the chance to work in an industry you're particularly interested in. Your work will then help a number of people within an organization, as well as assisting in the smooth operation of the business as a whole. For example, depending on their requirements, some companies may use the data for communications purposes, whereas some may use it for targeted marketing. To help non-technical people understand how to use your system and ensure it's used properly. You may also be writing reports and training manuals. You could also be involved in direct training, either of your own team or of employees that will use the system.

As a database manager, your daily tasks can also include:

- Overlooking database design
- Training and managing junior staff in your team
- Setting up and testing new database and data handling systems
- Monitoring database efficiency
- Designing and preparing reports for management
- Developing protocols for data processing
- Creating complex query definitions that allow data to be extracted
- Training colleagues in how to input and extract data

Definition - *Database Performance Monitoring*

Database performance monitoring is the act of measuring the performance of a given database in real time in order to determine problems and other factors that may cause problems in the future. It is also a good way to determine which areas of the database can be enhanced or optimized to increase efficiency and performance. This is usually done through monitoring software and tools either built in to the database management software or installed from third-party providers.

explains *Database Performance Monitoring*

Database performance monitoring is an active endeavor that is done to find problems and fix them, as well as to find ways of improving database performance. Through various tools and interfaces, a database administrator can monitor the performance of various portions of the database serving different applications and will be able to get visibility into the application's database performance through a specialized UI or a Web interface.
The main goal of database performance monitoring is to assess how a database server is performing, both hardware and software. This involves taking regular snapshots of performance indicators over time in order to determine the exact time that problems such as bottlenecks occur to be able to gain insight on what exactly caused the problems at that exact time and hopefully find a good solution. This is because most problems occur at different times, and it is simply not possible for one administrator to monitor the performance in real time all the time. So, looking at historical data is usually the approach to monitoring. Most monitoring tools have some sort of alarm and notification system to notify the administrator of ongoing problems.
The benefits of database performance monitoring are as follows:

- Determine whether performance can be improved
- Determine performance and security flaws by evaluating application and user activity
- Troubleshoot and debug problems in application components using the database

Database machine

A **database machines** or **back end processor** is a computer or special hardware that stores and retrieves data from a database. It is specially designed for database access and is coupled to the main (front-end) computer(s) by a high-speed channel. The database machine is tightly coupled to the main CPU, whereas the database server is loosely coupled via the network. Database machines can transfer large packets of data to the mainframe using hundreds to thousands of microprocessors with database software. The front end processor receives the data and displays it. The back end processor on the other hand analyzes and stores the data from the front end processor. Back end processors result in higher performance, increasing host main memory, increasing database recovery and security, and decrease of cost to manufacture. The database machine contrasts with a database server, which is a computer in a local area network that holds a database.

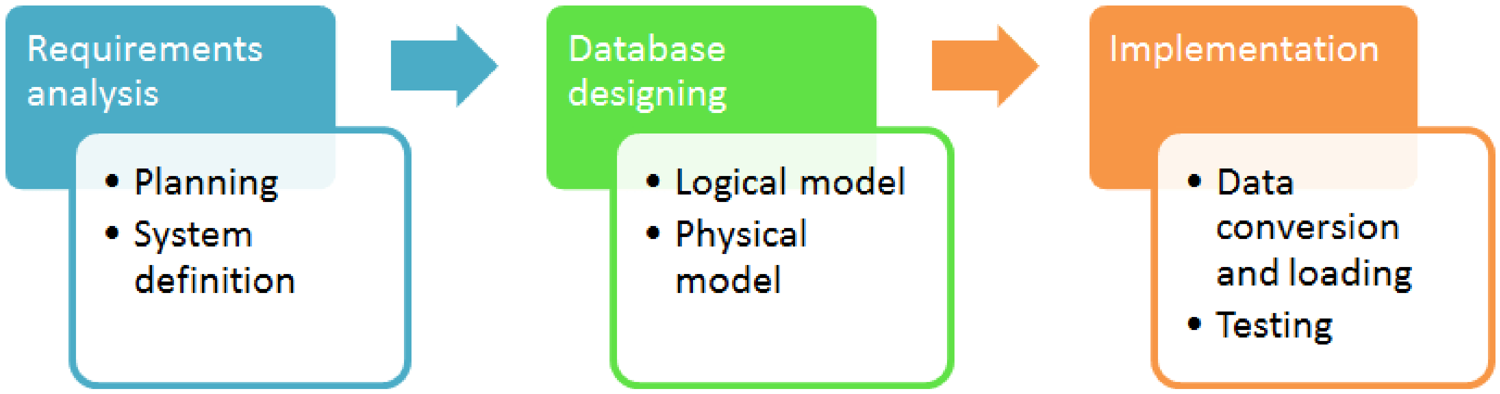
According to Julie A. McCann, "Currently a DBMS controls the organisation, storage and retrieval of data whilst regulating the security and integrity of the database, it accepts requests for data from the application programs and instructs the operating system (OS) to transfer the appropriate data."[1]

An example is the IBM System/38.

The design of RDBMS for an organization process consists of the following steps:

1. Determine the purpose of your **database**. ...

2. Find and organize the information required. ...
3. Divide the information into tables. ...
4. Turn information items into columns. ...
5. Specify primary keys. ...
6. Set up the table relationships. ...
7. Refine your **design**. ...
8. Apply the normalization rules.



OBJECT MODELING

In the object oriented data model (OODM), both data and their relationships are contained in a single structure known as an object.

In turn, the OODM is the basis for the object-oriented database management system (OODBMS).

The Components of the Object Oriented Data Model

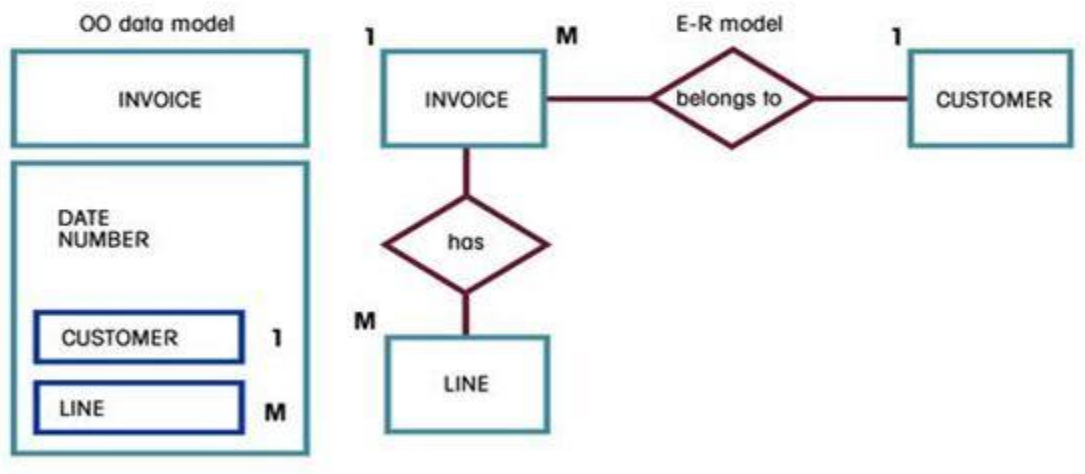
• An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity. (The object's semantic content is defined through several of the items in this list.)

• Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.

• Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's entity set. However, a class is different from an entity set in that it contains a set of procedures known as methods. A class's method represents a real-world action such as finding a selected PERSON's name, changing a PERSON's name, or printing a PERSON's address. In other words, methods are the equivalent of procedures in traditional programming languages. In OO terms, methods define an object's behavior.

• Classes are organized in a class hierarchy. The class hierarchy resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class. (Note the similarity to the hierarchical data model in this respect.)

• Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.



A comparison of OO data model and ER model

Advantages of Object Oriented Data Model

1. Add semantic content
2. Visual presentation includes semantic content
3. Database integrity
4. Both structural and data independence

Disadvantages of Object Oriented Data Model

1. Lack of OODM standards
2. Complex navigational data access
3. Steep learning curve

4. High system overhead slows transactions

perspective of data modeling

Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the **data** elements and the relationships between the **data** elements. **Data Models** are used to show how **data** is stored, connected, accessed and updated in the **database management system**

Unit -4

Concurrency Control in DBMS

Concurrency Control deals with **interleaved execution** of more than one transaction. In the next article, we will see what serializability is and how to find whether a schedule is serializable or not.

What is Transaction?

A set of logically related operations is known as transaction. The main operations of a transaction are:

Read(A): Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in main memory.

Write (A): Write operation Write(A) or W(A) writes the value back to the database from buffer.

Let us take a debit transaction from an account which consists of following operations:

1. R(A);
2. A=A-1000;
3. W(A);

Assume A's value before starting of transaction is 5000.

- The first operation reads the value of A from database and stores it in a buffer.
- Second operation will decrease its value by 1000. So buffer will contain 4000.
- Third operation will write the value from buffer to database. So A's final value will be 4000.

But it may also be possible that transaction may fail after executing some of its operations. The failure can be because of **hardware, software or power** etc. For example, if debit transaction discussed above fails after executing operation 2, the value of A will remain 5000 in the database which is not acceptable by the bank. To avoid this, Database has two important operations:

Commit: After all instructions of a transaction are successfully executed, the changes made by transaction are made permanent in the database.

Rollback: If a transaction is not able to execute all operations successfully, all the changes made by transaction are undone.

Properties of a transaction

Atomicity: As a transaction is set of logically related operations, **either all of them should be executed or none**. A debit transaction discussed above should either execute all three operations or none.If debit transaction fails after executing operation 1 and 2 then its new value 4000 will not be updated in the database which leads to inconsistency.

Consistency: If operations of debit and credit transactions on same account are executed concurrently, it may leave database in an inconsistent state.

- For Example, T1 (debit of Rs. 1000 from A) and T2 (credit of 500 to A) executing concurrently, the database reaches inconsistent state.

- Let us assume Account balance of A is Rs. 5000. T1 reads A(5000) and stores the value in its local buffer space. Then T2 reads A(5000) and also stores the value in its local buffer space.
- T1 performs A=A-1000 (5000-1000=4000) and 4000 is stored in T1 buffer space. Then T2 performs A=A+500 (5000+500=5500) and 5500 is stored in T2 buffer space. T1 writes the value from its buffer back to database.
- A's value is updated to 4000 in database and then T2 writes the value from its buffer back to database. A's value is updated to 5500 which shows that the effect of debit transaction is lost and database has become inconsistent.
- To maintain consistency of database, we need **concurrency control protocols** which will be discussed in next article. The operations of T1 and T2 with their buffers and database have been shown in Table 1.

T1	**T1's buffer space**	**T2**	**T2's Buffer Space**	**Database**
				A=5000
R(A);	A=5000			A=5000
	A=5000	R(A);	A=5000	A=5000
A=A-1000;	A=4000		A=5000	A=5000
	A=4000	A=A+500;	A=5500	
W(A);			A=5500	A=4000
		W(A);		A=5500

Isolation: Result of a transaction should not be visible to others before transaction is committed. For example, Let us assume that A's balance is Rs. 5000 and T1 debits Rs. 1000 from A. A's new balance will be 4000. If T2 credits Rs. 500 to A's new balance, A will become 4500 and after this T1 fails. Then we have to rollback T2 as well because it is using value produced by T1. So a transaction results are not made visible to other transactions before it commits.

Durable: Once database has committed a transaction, the changes made by the transaction should be permanent. e.g.; If a person has credited $500000 to his account, bank can't say that the update has been lost. To avoid this problem, multiple copies of database are stored at different locations.

What is a Schedule?

A schedule is a series of operations from one or more transactions. A schedule can be of two types:

- **Serial Schedule:** When one transaction completely executes before starting another transaction, the schedule is called serial schedule. A serial schedule is always consistent. e.g.; If a schedule S has debit transaction T1 and credit transaction T2, possible serial schedules are T1 followed by T2 (T1->T2) or T2 followed by T1 ((T1->T2). A serial schedule has low throughput and less resource utilization.
- **Concurrent Schedule:** When operations of a transaction are interleaved with operations of other transactions of a schedule, the schedule is called Concurrent schedule. e.g.; Schedule of debit and credit transaction shown in Table 1 is concurrent in nature. But concurrency can lead to inconsistency in the database. The above example of a concurrent schedule is also inconsistent.

Question: Consider the following transaction involving two bank accounts x and y:

1. read(x);
2. x := x – 50;
3. write(x);
4. read(y);
5. y := y + 50;
6. write(y);

The constraint that the sum of the accounts x and y should remain constant is that of?

1. Atomicity
2. Consistency
3. Isolation
4. Durability

[GATE 2015]

Solution: As discussed in properties of transactions, consistency properties says that sum of accounts x and y should remain constant before starting and after completion of transaction. So, the correct answer is B.

Concurrency Control Techniques

Concurrency control is provided in a database to:

- (i) enforce isolation among transactions.
- (ii) preserve database consistency through consistency preserving execution of transactions.
- (iii) resolve read-write and write-read conflicts.

Various concurrency control techniques are:

```
1. Two-phase locking Protocol
2. Time stamp ordering Protocol
3. Multi version concurrency control
4. Validation concurrency control
```

These are briefly explained below.

1. Two-Phase Locking Protocol:
Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.
The 3 activities taking place in the two phase update algorithm are:

```
(i). Lock Acquisition
(ii). Modification of Data
(iii). Release Lock
```

Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

- **(i) Growing Phase:**
  In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.
- **(ii) Shrinking Phase:**
  In this phase a transaction can only release locks but cannot acquire any.

2. Time Stamp Ordering Protocol:
A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.
The timestamp of a data item can be of 2 types:

- **(i) W-timestamp(X):**
  This means the latest time when the data item X has been written into.
- **(ii) R-timestamp(X):**
  This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

3. Multiversion Concurrency Control:
Multiversion schemes keep old versions of data item to increase concurrency.
Multiversion 2 phase locking:
Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

4. Validation Concurrency Control:
The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

- **(i)** During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
- **(ii)** During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, he transaction is restarted and the changes are discarded.
- **(iii)** During the write phase, the changes are permanently applied to the database.

Lock Based Concurrency Control Protocol

First things first, I hope you are familiar to some of the concepts relating to Transactions.

- What is a Recoverable Schedule?
- What are Cascading Rollbacks and Cascadeless schedules?
- Determining if a schedule is Conflict Serializable.

Now, we all know the four properties a transaction must follow. Yes, you got that right, I mean the **ACID** properties. Concurrency control techniques are used to ensure that the *Isolation* (or non-interference) property of concurrently executing transactions is maintained.
A trivial question I would like to pose in front of you, (I know you must know this but still) why do you think that we should have interleaving execution of transactions if it may lead to problems such as Irrecoverable Schedule, Inconsistency and many more threats.
Why not just let it be Serial schedules and we may live peacefully, no complications at all.
Yes, the performance effects the efficiency too much which is not acceptable.
Hence a Database may provide a mechanism that ensures that the schedules are either conflict or view serializable and recoverable (also preferably cascadeless). Testing for a schedule for Serializability after it has executed is obviously *too late!*
So we need Concurrency Control Protocols that ensures Serializability .
Concurrency-control protocols : allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless.
These protocols do not examine the precedence graph as it is being created, instead a protocol imposes a discipline that avoids non-seralizable schedules.
Different concurrency control protocols provide different advantages between the amount of concurrency they allow and the amount of overhead that they impose.
We'll be learning some protocols which are important for GATE CS. Questions from this

topic is frequently asked and it's recommended to learn this concept. (At the end of this series of articles I'll try to list all theoretical aspects of this concept for students to revise quickly and they may find the material in one place.) Now, let's get going:
Different categories of protocols:

- **Lock Based Protocol**
  - Basic 2-PL
  - Conservative 2-PL
  - Strict 2-PL
  - Rigorous 2-PL
- **Graph Based Protocol**
- **Time-Stamp Ordering Protocol**
- **Multiple Granularity Protocol**
- **Multi-version Protocol**

For GATE we'll be focusing on the First three protocols.

Lock Based Protocols –

A lock is a variable associated with a data item that describes a status of data item with respect to possible operation that can be applied to it. They synchronize the access by concurrent transactions to the database items. It is required in this protocol that all the data items must be accessed in a mutually exclusive manner. Let me introduce you to two common locks which are used and some terminology followed in this protocol.

1. **Shared Lock (S):** also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.
2. **Exclusive Lock (X):** Data item can be both read as well as written.This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.

Lock Compatibility Matrix –

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other
  transactions.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive(X) on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. Then the lock is granted.

Upgrade / Downgrade locks : A transaction that holds a lock on an item **A** is allowed under certain condition to change the lock state from one state to another.
Upgrade: A S(A) can be upgraded to X(A) if T_i is the only transaction holding the S-lock on element A.
Downgrade: We may downgrade X(A) to S(A) when we feel that we no longer want to write on data-item A. As we were holding X-lock on A, we need not check any conditions.

So, by now we are introduced with the types of locks and how to apply them. But wait, just by applying locks if our problems could've been avoided then life would've been so simple! If you have done Process Synchronization under OS you must be familiar with one consistent problem, starvation and Deadlock! We'll be discussing them shortly, but just so you know we have to apply Locks but they must follow a set of protocols to avoid such undesirable problems. Shortly we'll use 2-Phase Locking (2-PL) which will use the concept of Locks to avoid deadlock. So, applying simple locking, we may not always produce Serializable results, it may lead to Deadlock Inconsistency.

Problem With Simple Locking…

Consider the Partial Schedule:

	T_1	T_2
1	lock-X(B)	
2	read(B)	
3	B:=B-50	
4	write(B)	
5		lock-S(A)
6		read(A)
7		lock-S(B)
8	lock-X(A)	
9	……	……

Deadlock – consider the above execution phase. Now, **T_1** holds an Exclusive lock over B, and **T_2** holds a Shared lock over A. Consider Statement 7, **T_2** requests for lock on B, while in Statement 8 **T_1** requests lock on A. This as you may notice imposes a **Deadlock** as none can proceed with their execution.
Starvation – is also possible if concurrency control manager is badly designed. For example: A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item. This may be avoided if the concurrency control manager is properly designed.
Phew… I hope you are now familiar with why we should study Concurrency Control Protocols. Moreover, you should be familiar with basics of Lock Based Protocols and problems with Simple Locking.

Introduction of Relational Model and Codd's Rules in Relational DBMS

Terminology

Relational Model: Relational model represents data in the form of relations or tables.
Relational Schema: Schema represents structure of a relation. e.g.; Relational Schema of STUDENT relation can be represented as:
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE)
Relational Instance: The set of values present in a relation at a particular instance of time is known as relational instance as shown in Table 1 and Table 2.
Attribute: Each relation is defined in terms of some properties, each of which is known as attribute. For Example, STUD_NO, STUD_NAME etc. are attributes of relation STUDENT.
Domain of an attribute: The possible values an attribute can take in a relation is called its domain. For Example, domain of STUD_AGE can be from 18 to 40.
Tuple: Each row of a relation is known as tuple. e.g.; STUDENT relation given below has 4 tuples.
NULL values: Values of some attribute for some tuples may be unknown, missing or undefined which are represented by NULL. Two NULL values in a relation are considered different from each other.
Table 1 and Table 2 represent relational model having two relations STUDENT and STUDENT_COURSE.

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Codd Rules

Codd rules were proposed by E.F. Codd which should be satisfied by relational model.

1. **Foundation Rule:** For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.
2. **Information Rule:** Data stored in Relational model must be a value of some cell of a table.

3. **Guaranteed Access Rule:** Every data element must be accessible by table name, its primary key and name of attribute whose value is to be determined.
4. **Systematic Treatment of NULL values:** NULL value in database must only correspond to missing, unknown or not applicable values.
5. **Active Online Catalog:** Structure of database must be stored in an online catalog which can be queried by authorized users.
6. **Comprehensive Data Sub-language Rule:** A database should be accessible by a language supported for definition, manipulation and transaction management operation.
7. **View Updating Rule:** Different views created for various purposes should be automatically updatable by the system.
8. **High level insert, update and delete rule:** Relational Model should support insert, delete, update etc. operations at each level of relations. Also, set operations like Union, Intersection and minus should be supported.
9. **Physical data independence:** Any modification in the physical location of a table should not enforce modification at application level.
10. **Logical data independence:** Any modification in logical or conceptual schema of a table should not enforce modification at application level. For example, merging of two tables into one should not affect application accessing it which is difficult to achieve.
11. **Integrity Independence:** Integrity constraints modified at database level should not enforce modification at application level.
12. **Distribution Independence:** Distribution of data over various locations should not be visible to end-users.
13. **Non-Subversion Rule:** Low level access to data should not be able to bypass integrity rule to change data.

Data integrity

Definition - *Data Integrity*

Data integrity is the overall completeness, accuracy and consistency of data. This can be indicated by the absence of alteration between two instances or between two updates of a data record, meaning data is intact and unchanged. Data integrity is usually imposed during the database design phase through the use of standard procedures and rules. Data integrity can be maintained through the use of various error-checking methods and validation procedures.

explains *Data Integrity*

Data integrity is enforced in both hierarchical and relational database models. The following three integrity constraints are used in a relational database structure to achieve data integrity:

- **Entity Integrity:** This is concerned with the concept of primary keys. The rule states that every table must have its own primary key and that each has to be unique and not null.
- **Referential Integrity**: This is the concept of foreign keys. The rule states that the foreign key value can be in two states. The first state is that the foreign key value would refer to a primary key value of another table, or it can be null. Being null could simply mean that there are no relationships, or that the relationship is unknown.
- **Domain Integrity:** This states that all columns in a relational database are in a defined domain.

The concept of data integrity ensures that all data in a database can be traced and connected to other data. This ensures that everything is recoverable and searchable. Having a single, well-defined and well-controlled data integrity system increases stability, performance, reusability and maintainability. If one of these features cannot be implemented in the database, it must be implemented through the software.

Data redundancy

Data **redundancy** is a condition created within a database or data storage technology in which the same piece of data is held in two separate places. ... Whenever data is repeated, this basically constitutes data **redundancy**. This can occur by accident, but is also done deliberately for backup and recovery purposes.

Definition - *Data Redundancy*

Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places.

This can mean two different fields within a single database, or two different spots in multiple software environments or platforms. Whenever data is repeated, this basically constitutes data redundancy. This can occur by accident, but is also done deliberately for backup and recovery purposes.

explains *Data Redundancy*

Within the general definition of data redundancy, there are different classifications based on what is considered appropriate in database management, and what is considered excessive or wasteful. Wasteful data redundancy generally occurs when a given piece of data does not have to be repeated, but ends up being duplicated due to inefficient coding or process complexity.

A positive type of data redundancy works to safeguard data and promote consistency. Many developers consider it acceptable for data to be stored in multiple places. The key is to have a central, master field or space for this data, so that there is a way to update all of the places where data is redundant through one central access point. Otherwise, data redundancy can lead to big problems with data inconsistency, where one update does not automatically update another field. As a result, pieces of data that are supposed to be identical end up having different values.

Primary key and foreign key are the two most important and common types of **keys** used in relational databases. A **primary key** is a special **key** used to uniquely identify records in a table, whereas a **foreign key** is used to establish relationship between two tables.

What is the Difference between a Primary Key and a Foreign Key?

Before we can dig into the difference, let's first explore primary and foreign key characteristics. Let's start out by learning about primary keys.

Primary Keys

In order for a table to qualify as a relational table it must have a primary key.

The **primary key** consists of one or more columns whose data contained within is used to uniquely identify each row in the table. You can think of them as an address. If the rows in a table were mailboxes, then the primary key would be the listing of street addresses.
When a primary key is composed of multiple columns, the data from each column is used to determine whether a row is unique.

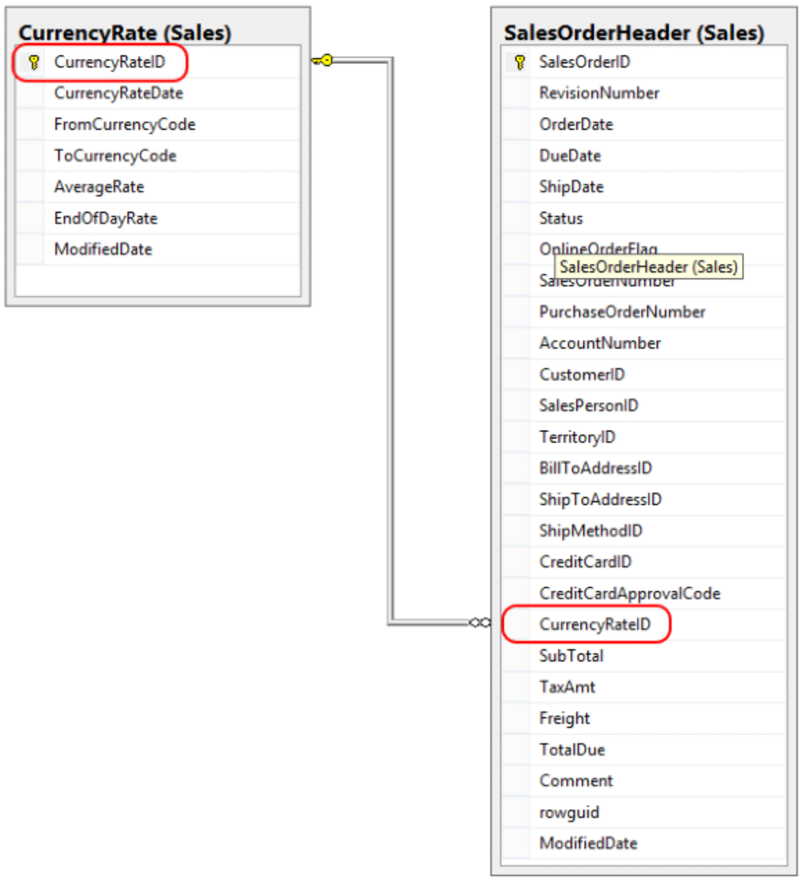
In order to be a primary key, several conditions must hold true. First, as we mentioned, the columns must be unique. To clarify, we're referring to the data within the rows, not the column names themselves. Also, no value in the columns can be blank or NULL.
When defining a table you specify the primary key. A table has just one primary key, and its definition is mandatory.

The primary key for each table is stored in an index. The index is used to enforce the uniqueness requirement. It also makes it easy for foreign key values to refer back to corresponding primary key values, as we will learn about in the following section.

Foreign Keys

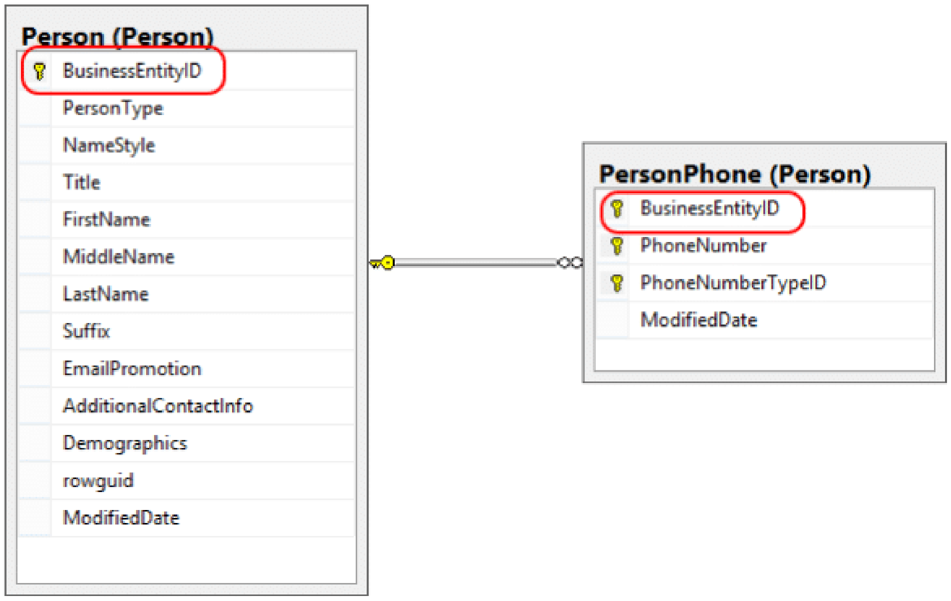
A **foreign key** is a set of one or more columns in a table that refers to the primary key in another table. There isn't any special code, configurations, or table definitions you need to place to officially "designate" a foreign key.

In the diagram below look at the SalesOrderHeader table. The column SalesOrderHeader.CurrencyRateID is a foreign key since it is related to the CurrencyRate.CurrencyRateID. This column CurrencyRate.CurrencyRateID is the primary key of the CurrencyRate table.



Foreign Keys as Part of Primaries

Look at the following diagram. Which column is the foreign key?



If you said it was PersonPhone.BusinessEntityID then you are correct. The reason it is a foreign key is that it is referring to a primary key, Person.BusinessEntityID, in the other table.

Coincidentally, PersonPhone.BusinessEntityID is not only a foreign key, but is also part of PersonPhone's primary key. The PersonPhone table's primary key is the combination of BusinessEntityID, PhoneNumer, and PhoneNumberTypeID.

I agree this is confusing, but it is allowed and not a bad practice.

Unlike primary keys, foreign keys can contain duplicate values. Also, it is OK for them contain NULL values.

Foreign Key Constraints

Some database management systems, such as SQL Server allow you to set up foreign key constraints. These help to enforce referential integrity. In their simplest form, a foreign key constraint stops you from entering values that aren't found in the related table's primary key.

Using the first diagram as our example, you can't enter SalesOrderHeader.CurrencyRateID if it doesn't already exist in the CurrencyRate table.

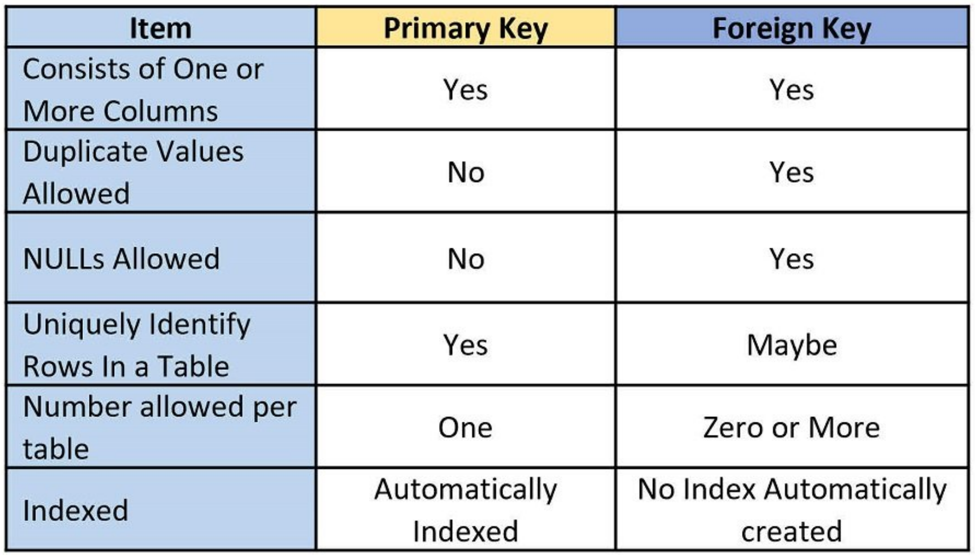
These constraints come into effect in several ways:

1. They bar you from changing the foreign key value to one which doesn't exist as a value in the related table's primary key.
2. They stop you from deleting a row from the primary key table. This stops you from creating orphan records. Orphan records are described as "child records with no parents."
3. They stop you from adding a foreign key value that doesn't exist in the primary key.

In summary, the constraints enforce the relationship between the primary and foreign key tables.

Comparison of Primary Keys to Foreign Keys

To summarize here is a comparison of Primary to Foreign Keys



Item	Primary Key	Foreign Key
Consists of One or More Columns	Yes	Yes
Duplicate Values Allowed	No	Yes
NULLs Allowed	No	Yes
Uniquely Identify Rows In a Table	Yes	Maybe
Number allowed per table	One	Zero or More
Indexed	Automatically Indexed	No Index Automatically created

Unit-5

Object Database Management

An object database management system (ODBMS, also referred to as object-oriented database management system or OODBMS), is a database management system (DBMS) that supports the modeling and creation of data as objects. This includes some kind of support for classes of objects and the inheritance of class properties and methods by subclasses and their objects.

Definition and Overview of ODBMS

The **ODBMS** which is an abbreviation for **object oriented database management system**, is the data model in which data is stored in form of objects, which are instances of classes. These classes and objects together makes an object oriented data model.

Components of Object Oriented Data Model:

The OODBMS is based on three major components, namely: Object structure, Object classes, and Object identity. These are explained as following below.

1. Object Structure:

The structure of an object refers to the properties that an object is made up of. These properties of an object are referred to as an attribute. Thus, an object is a real world world entity with certain attributes that makes up the object structure. Also an object encapsulates the data code into a single unit which in turn provides data abstraction by hiding the implementation details from the user.

The object structure is further composed of three types of components: Messages, Methods, and Variables. These are explained as following below.

1. **Messages –**
   A message provides an interface or acts as a communication medium between an object and the outside world. A message can be of two types:
   - **Read-only message:** If the invoked method does not change the value of a variable, then the invoking message is said to be a read-only message.
   - **Update message:** If the invoked method changes the value of a variable, then the invoking message is said to be an update message.
2. **Methods –**
   When a message is passed then the body of code that is executed is known as a method. Every time when a method is executed, it returns a value as output. A method can be of two types:
   - **Read-only method:** When the value of a variable is not affected by a method, then it is known as read-only method.
   - **Update-method:** When the value of a variable changes by a method, then it is known as an update method.
3. **Variables –**
   It stores the data of an object. The data stored in the variables makes the object distinguishable from one another.

2. Object Classes:
An object which is a real world entity is an instance of a class. Hence first we need to define a class and then the objects are made which differ in the values they store but share the same class definition. The objects in turn corresponds to various messages and variables stored in it.
Example –
```
class CLERK

  { //variables
     char name;
     string address;
     int id;
     int salary;

    //messages
     char get_name();
     string get_address();
     int annual_salary();
  };
```
In above example we can see, CLERK is a class that holds the object variables and messages.

An OODBMS also supports inheritance in an extensive manner as in a database there may be many classes with similar methods, variables and messages. Thus, the concept of class hierarchy is maintained to depict the similarities among various classes.

The concept of encapsulation that is the data or information hiding is also supported by object oriented data model. And this data model also provides the facility of abstract data types apart from the built-in data types like char, int, float. ADT's are the user defined data types that hold the values within it and can also have methods attached to it.

Database Design

students(student_id, student_name, address)
enrollment(student_id, subject_id, mark)
subject(subject_id, subject_name, department)

Create student table
```
CREATE TABLE STUDENTS
(
STUDENT_ID VARCHAR2(5) PRIMARY KEY,
STUDENT_NAME VARCHAR2(15),
    ADDRESS VARCHAR2(25)
);
```
Create enrollment table

```
CREATE TABLE ENROLMENT
(
STUDENT_ID VARCHAR2(5) REFERENCES STUDENTS(STUDENT_ID),
SUBJECT_ID VARCHAR2(6) REFERENCES SUBJECTS(SUBJECT_ID),
MARK NUMBER(3),
PRIMARY KEY(STUDENT_ID,SUBJECT_ID)
);
```

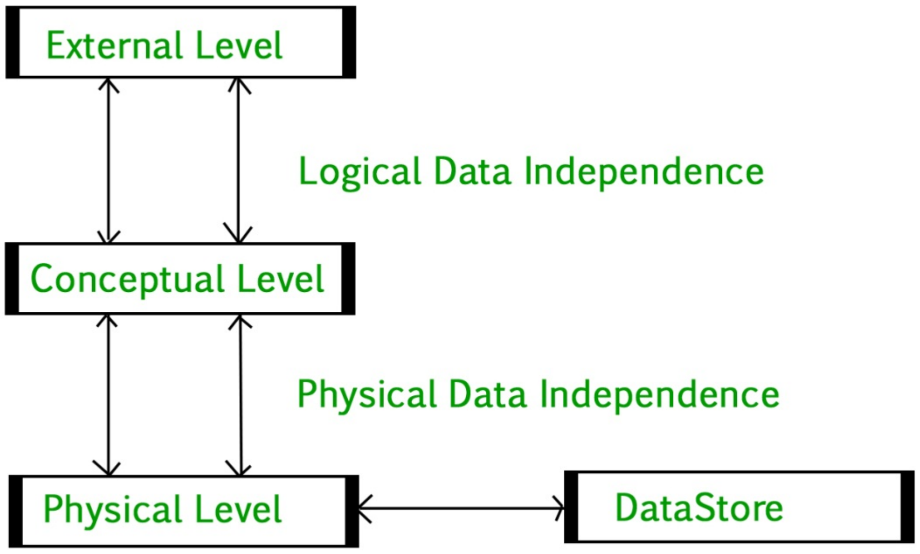
Create subject table

```
CREATE TABLE SUBJECTS
(
SUBJECT_ID VARCHAR2(6) PRIMARY KEY,
SUBJECT_NAME VARCHAR2(20),
DEPARTMENT VARCHAR2(20)
);
```

This is the skipped topic from architecture of RDBMS

RDBMS 3-tier Architecture

DBMS 3-tier architecture divides the complete system into three inter-related but independent modules as shown below:



1. **Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. Various users of DBMS are unaware of the locations of these objects.In simple terms,physical level of a database describes how the data is being stored in secondary storage devices like disks and tapes and also gives insights on additional storage details.

2. **Conceptual Level:** At conceptual level, data is represented in the form of various database tables. For Example, STUDENT database may contain STUDENT and COURSE tables which will be visible to users but users are unaware of their storage.Also referred as logical schema,it describes what kind of data is to be stored in the database.
3. **External Level:** An external level specifies a view of the data in terms of conceptual level tables. Each external level view is used to cater to the needs of a particular category of users. For Example, FACULTY of a university is interested in looking course details of students; STUDENTS are interested in looking at all details related to academics, accounts, courses and hostel details as well. So, different views can be generated for different users. The main focus of external level is data abstraction.

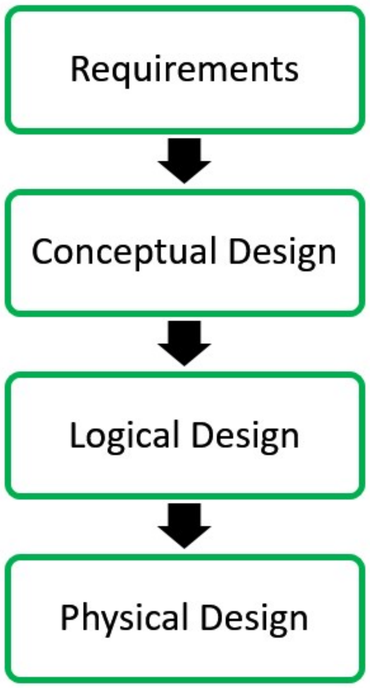
Data Independence

Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

1. **Physical Data Independence:** Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.
2. **Conceptual Data Independence:** The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user's view of the table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in conceptual schema are reflected in the user's view.

Phases of database design

Database designing for a real-world application starts from capturing the requirements to physical implementation using DBMS software which consists of following steps shown below:



Conceptual Design: The requirements of database are captured using high level conceptual data model. For Example, the ER model is used for the conceptual design of the database.
Logical Design: Logical Design represents data in the form of relational model. ER diagram produced in the conceptual design phase is used to convert the data into the Relational Model.
Physical Design: In physical design, data in relational model is implemented using commercial DBMS like Oracle, DB2.

Advantages of DBMS

DBMS helps in efficient organization of data in database which has following advantages over typical file system:

- **Minimized redundancy and data inconsistency:** Data is normalized in DBMS to minimize the redundancy which helps in keeping data consistent. For Example, student information can be kept at one place in DBMS and accessed by different users. This minimized redundancy is due to primary key and foreign keys
- **Simplified Data Access:** A user need only name of the relation not exact location to access data, so the process is very simple.
- **Multiple data views:** Different views of same data can be created to cater the needs of different users. For Example, faculty salary information can be hidden from student view of data but shown in admin view.
- **Data Security:** Only authorized users are allowed to access the data in DBMS. Also, data can be encrypted by DBMS which makes it secure.
- **Concurrent access to data:** Data can be accessed concurrently by different users at same time in DBMS.

- **Backup and Recovery mechanism:** DBMS backup and recovery mechanism helps to avoid data loss and data inconsistency in case of catastrophic failures.

Difference between RDBMS and OODBMS

RDBMS:
RDBMS stands for Relational Database Management System. It is a database management system based on the relational model i.e. the data and relationships are represented by a collection of inter-related tables. It is a DBMS that enables the user to create, update, administer and interact with a relational database. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
OODBMS:
OODBMS stands for Object-Oriented Database Management System. It is a DBMS where data is represented in the form of objects, as used in object-oriented programming. OODB implements object-oriented concepts such as classes of objects, object identity, polymorphism, encapsulation, and inheritance. An object-oriented database stores complex data as compared to relational database. Some examples of OODBMS are Versant Object Database, Objectivity/DB, ObjectStore, Caché and ZODB.
Difference between RDBMS and OODBMS:

BASIS	RDBMS	OODBMS
Long Form	Stands for Relational Database Management System.	Stands for Object Oriented Database Management System.
Way of storing data	Stores data in Entities, defined as tables hold specific information.	Stores data as Objects.
Data Complexity	Handles comparatively simpler data.	Handles larger and complex data than RDBMS.
Grouping	Entity type refers to the collection of entity that shares a common	Class describes a group of objects that have common relationships, behaviors, and also have similar

BASIS	RDBMS	OODBMS
	definition.	properties.
Data Handling	RDBMS stores only data.	Stores data as well as methods to use it.
Main Objective	Data Independence from application program.	Data Encapsulation.
Key	A Primary key distinctively identifies an object in a table.	An object identifier (OID) is an unambiguous, long-term name for any type of object or entity.

Difference between MySQL and MS SQL Server

SQL is an acronym for Structured Query Language. It is used to access, manipulate and retrieve information from a database.
MySQL is an open source Relational Database Management System (RDBMS) based on Structured Query Language (SQL). It runs on platforms like Linux, UNIX and Windows.
SQL Server is owned and developed by Microsoft Corporation. The primary function of SQL Server is the storage and access of data as it is required by other applications, whether they are running on other computers that are connected to a network, or the computer on which the server is stored.

MS SQL SERVER	MYSQL
Developed by Microsoft.	Developed by Oracle.
It supports programming languages like C++, JAVA, Ruby, Visual Basic, Delphi, R etc.	MySQL offers extended running support for languages like Perl, Tcl, Haskey etc.
Expects a large amount of operational storage space.	Expects less amount of operational storage space.
It enables for stopping query execution.	It doesn’t allow query cancellation mid-way

MS SQL SERVER	MYSQL
	in the process.
Doesn’t block the database while backing up the data.	Blocks the database while backing up the data.
It is not free.	It is open source. It is freely available.
It is a highly secured and doesn’t allow any kind of database file manipulation while running.	It allows database file manipulation while running.
It is available in multiple editions, such as Enterprise, Standard, Web, Workgroup, or Express.	It is available in MySQL Standard Edition, MySQL Enterprise Edition, and MySQL Cluster Grade Edition.

Structured Query Language (SQL)

Structured Query Language is a standard Database language which is used to create, maintain and retrieve the relational database. Following are some interesting facts about SQL.

- SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.
- We can write comments in SQL using “–” (double hyphen) at the beginning of any line.
- SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL
- Although there is an ISO standard for SQL, most of the implementations slightly vary in syntax. So we may encounter queries that work in SQL Server but do not work in MySQL.

MySQL

MySQL is an open-source relational database management system (RDBMS). It is the most popular database system used with PHP. MySQL is developed, distributed, and supported by Oracle Corporation.

- The data in a MySQL database are stored in tables which consists of columns and rows.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.

- MySQL is very fast, reliable, and easy to use database system.It uses standard SQL
- MySQL compiles on a number of platforms.

Difference between ODBC and JDBC

ODBC	JDBC
ODBC Stands for Open Database Connectivity.	JDBC Stands for java database connectivity.
Introduced by Microsoft in 1992.	Introduced by SUN Micro Systems in 1997.
We can use ODBC for any language like C,C++,Java etc.	We can use JDBC only for Java languages.
We can choose ODBC only windows platform.	We can Use JDBC in any platform.
Mostly ODBC Driver developed in native languages like C,C++.	JDBC Stands for java database connectivity.
For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent.	For Java application it is highly recommended to use JDBC because there we no performance & platform dependent problem.
ODBC is procedural.	JDBC is object oriented.

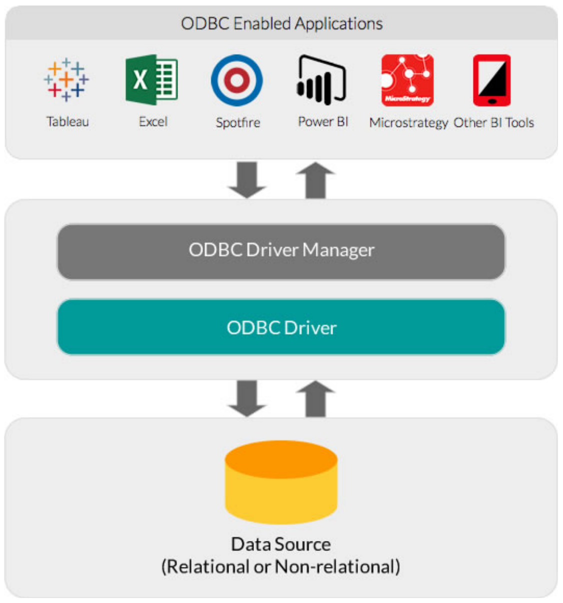
ODBC

Stands for "Open Database Connectivity." With all the different types of databases available, such as Microsoft Access, File maker, and MySQL, it is important to have a standard way of transferring data to and from each kind of database. For this reason, the SQL Access group created the ODBC standard back in 1992. Any application that supports ODBC can access information from an ODBC-compatible database, regardless of what database management system the database uses.

For a database to be ODBC-compatible, it must include an ODBC database driver. This allows other applications to connect to and access information from the database with a standard set of commands. The driver translates standard ODBC commands into commands understood by the database's proprietary system. Thanks to ODBC, a single application (such as Web server program) can access information from several different databases using the same set of commands.

ODBC Architecture

The architecture of ODBC-based data connectivity is as follows:



ODBC Enabled Application

This is any ODBC compliant application, such as Microsoft Excel, Tableau, Crystal Reports, Microsoft Power BI, or similar application (Spreadsheet, Word processor, Data Access & Retrievable Tool, etc.). The ODBC enabled application performs processing by passing SQL Statements to and receiving results from the ODBC Driver Manager.

ODBC Driver Manager

The ODBC Driver Manager loads and unloads ODBC drivers on behalf of an application. The Windows platform comes with a default Driver Manager, while non-windows platforms have the choice to use an open source ODBC Driver Manager like unixODBC and iODBC. The ODBC Driver Manager processes ODBC function calls, or passes them to an ODBC driver and resolves ODBC version conflicts.

ODBC Driver

The ODBC driver processes ODBC function calls, submits SQL requests to a specific data source and returns results to the application. The ODBC driver may also modify an application’s request so that the request conforms to syntax supported by the associated database

Data Source

A data source is simply the source of the data. It can be a file, a particular database on a DBMS, or even a live data feed. The data might be located on the same computer as the program, or on another computer somewhere on a network.

Unit-6

What is middleware?

Middleware is software which lies between an operating system and the applications running on it. Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications. It is sometimes called plumbing, as it connects two applications together so data and databases can be easily passed between the “pipe.” Using middleware allows users to perform such requests as submitting forms on a web browser or allowing the web server to return dynamic web pages based on a user’s profile.

Common middleware examples include database middleware, application server middleware, message-oriented middleware, web middleware and transaction-processing monitors. Each programme typically provides messaging services so that different applications can communicate using messaging frameworks like simple object access protocol (SOAP), web services, representational state transfer (REST) and JavaScript object notation (JSON). While all middleware performs communication functions, the type a company chooses to use will depend on what service is being used and what type of information needs to be communicated. This can include security authentication, transaction management, message queues, applications servers, web servers and directories. Middleware can also be used for distributed processing with actions occurring in real time rather than sending data back and forth.

Work of middleware

Middleware steps in to provide a unified means for all these systems to communicate and interact with each other.

To do this, a lot of middleware is cross-language, which means that it is capable of understanding and processing several different operating languages, such as Ruby on Rails, Java, C++, PHP, and so on.

Apart from allowing communication between fundamentally different systems, middleware also performs several other functions.

These include:

- Hiding the distributed nature of an application. On the surface, applications appear to be one unified package. Below the surface, however, they are comprised of several interconnected elements running in distributed locations. Middleware helps these different elements work in harmony to provide a unified experience for the user, despite the distributed nature of the application.

- Hiding the heterogeneity of the enterprise. The enterprise is usually made up of different hardware, different operating systems and different communication protocols. Middleware allows these different systems to work together while masking their differences.
- Providing application developers with uniform and standard high-level enterprises that they can use to build applications that can be run on different hardware and operating systems and work with each other.
- Avoiding duplication and enabling compatibility between applications by providing a common framework for performing various general purpose functions.

All in all, middleware helps make application development easier. In order to support application development, middleware uses the following components:

- **Database software**: Most multi-tier systems will require a database. Middleware acts as the link between the client and the server. It accepts client requests, passes them on to the database server and then passes back the response to the client.
- **Application server:** This is the part of the application that holds the business logic of the application.
- **Portal**: This is an interaction tool that is used to provide a selected audience with access to business applications, relevant information, instant messaging, discussion forums, and other company resources.
- **Service Oriented Architecture (SOA):** This is a framework that is used to easily design, develop and deploy applications. Many types of middleware use SOA with prebuilt services that can be utilized by multiple systems.
- **Web server**: The role of the web server is to process and deliver client requests. Web servers provide one of the best and most flexible options for the integration of different systems.

USES OF MIDDLEWARE

We have already seen that middleware acts as a link between different software.

But what exactly does it do apart from acting as a conduit between the different software?

Some of the uses of middleware include:

- **Transaction management**: Middleware can be used to manage and control individual transactions and ensure that any problems do not corrupt the system or database.
- **Application server**: Middleware can be used to host an API, allowing other applications to access and use the main application’s business logic and processes.
- **Security**: Middleware can be used to authenticate client programs and confirm that the program and the user behind the program are actually who they claim to be.
- **Message queues**: Middleware can be used to pass messages between different systems or software. The messages can then trigger a transaction or other action.

- **Directory**: Middleware can be used as a directory, enabling client programs to locate other services within a distributed enterprise.
- **Web server:** Middleware can also accept client requests from web browsers and channel them to the main server/database and then deliver the responses to the browsers.

TYPES OF MIDDLEWARE

As I mentioned earlier, the term middleware can be used to refer to any software that sits between two different applications.

As such, there is a wide variety of what counts as middleware.

However, middleware can still be classified into broad categories depending on their particular function. Some of the common types of middleware include:

Message Oriented Middleware (MOM)

This is software infrastructure that allows messages to be sent and received over distributed applications.

Message oriented middleware is one of the most widely used types of middleware. With message oriented middleware, it becomes less complicated to use applications spread over various platforms and working across various operating systems and network protocols.

In addition to enabling the transmission of messages across distributed applications, message oriented middleware also has a queuing mechanism that allows the interaction between the server and the client to happen metachronously in situations where the target node is busy or slow.

This prevents the message from getting misplaced while awaiting to get to the server or client. An example of message oriented middleware is email systems.

Remote Procedure Call (RPC) Middleware

This is a client-server interaction that makes it possible for the functionality of an application to be distributed across multiple platforms.

This type of middleware is a protocol that is used by a local program to request a service from a program located on a remote computer without having particulars of network details.

This type of middleware is most used to execute synchronous data transfers, where the both the client and the server need to be online at the time of the communication.

Database Middleware

This type of middleware allows direct access and interaction with a database. Database middleware is the most common and most widely used type of middleware.

It is mostly used by developers as a mechanism to request information from a database hosted either locally or remotely.

A good example of database middleware is the SQL database software.

Application Programming Interface (API)

An API is a set of protocols, tools and definitions for building applications, which allow a secondary application or service to communicate with a primary application or service, without having to know how the primary application or service is being implemented.

Object Middleware

Also known as an object request broker, the role of object middleware is to control the communication between objects in distributed computing.

Object middleware allows one computer to make program calls to another through a computer network.

It also allows requests and objects to be sent through an object-oriented system.

Transaction Processing (TP) Middleware

This is a type of middleware whose role is to reinforce the function of electronic transactions.

Transactional middleware does this by controlling transaction apps, pushing database updates related to the transaction and enforcing the business rules and logic of the transaction.

Robotic Middleware

This type of middleware is very handy when it comes building extensive software systems for controlling robot systems.

Robotic middleware helps to manage and control the heterogeneity and complexity of the hardware and software systems that form part of a robot.

Integration Middleware

This type of middleware provides an integration framework through which operations, executions and runtime services from several apps can be monitored and controlled.

Integration middleware can also be useful in combining data from several different sources into one unified platform where users can access and manipulate the data.

Application Framework

This is a framework that provides the basic structure on which applications for a particular environment can be built.

The application framework acts as a backbone that supports the application. It also provides a server on which the application will run.

Using an application framework makes the application development process a lot simpler.

Device Middleware

This is a type of middleware that provides a set of tools which are used to build applications meant to be run in a specific hardware environment.

Game Engines

This type of middleware provides game designers with access to tools that make the game creation process easier.

Game engines utilize tools such as game scripting, physics simulations, and graphics rendering.

Portals

Though they might not actually be a type of middleware, enterprise portal servers are also sometimes referred to middleware because they enable a smooth front-end integration.

The main role of portals is to allow interaction between a client device and back end systems.

Content-Centric Middleware

This is a type of middleware that makes it possible for developers to extract some piece of content without having to know how the system obtains the content.

This type of middleware is commonly used in most content-oriented web-based applications.

IMPORTANCE OF MIDDLEWARE

If you company relies a lot on data, then you might consider implementing middleware so that you can integrate the data across various applications and systems.

Integration makes the flow of data across the various applications a lot easier and allows your company to focus on other important aspects of your business, since you no longer have to spend time on manual processes.

Some of the benefits to be gained by implementing middleware include:

Improved Agility

Today, businesses need to deliver services to customers across various platforms, including on the cloud, on mobile and through traditional application platforms.

Delivering services across all these platforms can be a challenge.

At the same time, customers expect a similar user experience regardless of the platform through which they access services.

In order to be able to deliver a seamless user experience, your company's IT landscape needs to be very agile.

Middleware can help provide this agility. It provides a framework that allows changes to be easily made to business processes.

This way, the business can easily respond and adapt to customer requirements and expectations and deliver new services much faster.

Increased Efficiency

Middleware technology is very useful when it comes to automating business processes.

With middleware, processes such as ordering and product configuration can be automated, leading to time and cost improvements when compared to performing these processes manually.

Members of staff who previously performed these processes can be deployed to other tasks.

Process automation also reduces the delivery time and makes customer interactions a lot simpler, ultimately increasing the total volume of business.

Products and services can be brought to market much faster without an increase in development costs.

For instance, one hotel chain used middleware technology from Oracle to provide users with real-time information about room availability and rates directly on Google Maps.

Portability and Reusability

While the advanced technology applications being released today make it easier to quickly make changes to business processes, some organizations may still rely on some old legacy organizations.

Implementing changes on these old legacy applications can be quite challenging.

However, middleware technology can be used to make these old systems more adaptable to changes.

This makes it easier to use older systems for new applications.

Additionally, middleware technology is very reusable, since it contains common components that can be utilized in multiple environments.

Because of this, an application built on top of certain middleware can be used on multiple platforms, making the application a lot more portable.

Cost Effectiveness

Due to the use of common components, developing applications on top of middleware technology means less effort is required to build the application from scratch.

This can result in significant reductions in both development time and project cost.

In addition, the use of middleware to automate business processes can also lead to significant cost savings.

Information Management

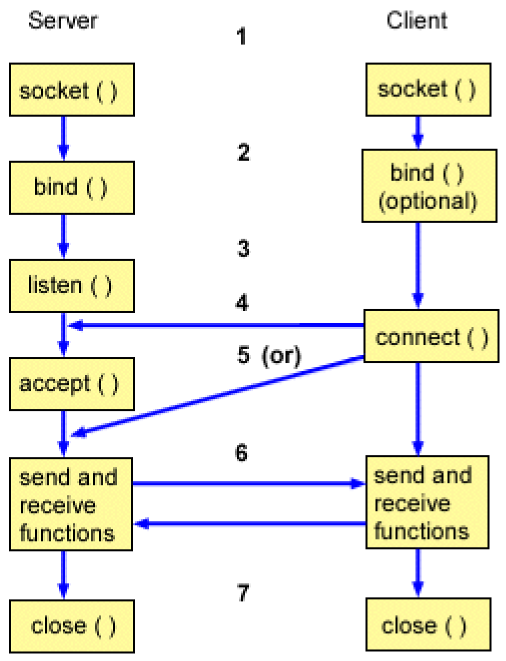
Information management is a crucial aspect of any large organization.

Middleware technology can make information management much easier by providing a framework on which an information management system can be designed, built and deployed.

Sockets

Sockets are commonly used for client and server interaction. ... The clients connect to the server, exchange information, and then disconnect. A **socket** has a typical flow of events. In a connection-oriented client-to-server model, the **socket** on the server process waits for requests from a client
Sockets are commonly used for client and server interaction. Typical system configuration places the server on one machine, with the clients on other machines. The clients connect to the server, exchange information, and then disconnect.

A socket has a typical flow of events. In a connection-oriented client-to-server model, the socket on the server process waits for requests from a client. To do this, the server first establishes (binds) an address that clients can use to find the server. When the address is established, the server waits for clients to request a service. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client. The following figure shows the typical flow of events (and the sequence of issued APIs) for a connection-oriented socket session. An explanation of each event follows the figure.



This is a typical flow of events for a connection-oriented socket:

1. The socket() API creates an endpoint for communications and returns a socket descriptor that represents the endpoint.
2. When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
3. The listen() API indicates a willingness to accept client connection requests. When a listen() API is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() API and the bind() API binds a name to the socket. A listen() API must be issued before an accept() API is issued.
4. The client application uses a connect() API on a stream socket to establish a connection to the server.
5. The server application uses the accept() API to accept a client connection request. The server must issue the bind() and listen() APIs successfully before it can issue an accept() API.
6. When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer APIs. Clients and servers have

many data transfer APIs from which to choose, such as send(), recv(), read(), write(), and others.

7. When a server or client wants to stop operations, it issues a close() API to release any system resources acquired by the socket.

Sockets some common characteristics.

- A socket is represented by an integer. That integer is called a *socket descriptor*.
- A socket exists as long as the process maintains an open link to the socket.
- You can name a socket and use it to communicate with other sockets in a communication domain.
- Sockets perform the communication when the server accepts connections from them, or when it exchanges messages with them.
- You can create sockets in pairs (only for sockets in the AF_UNIX address family).

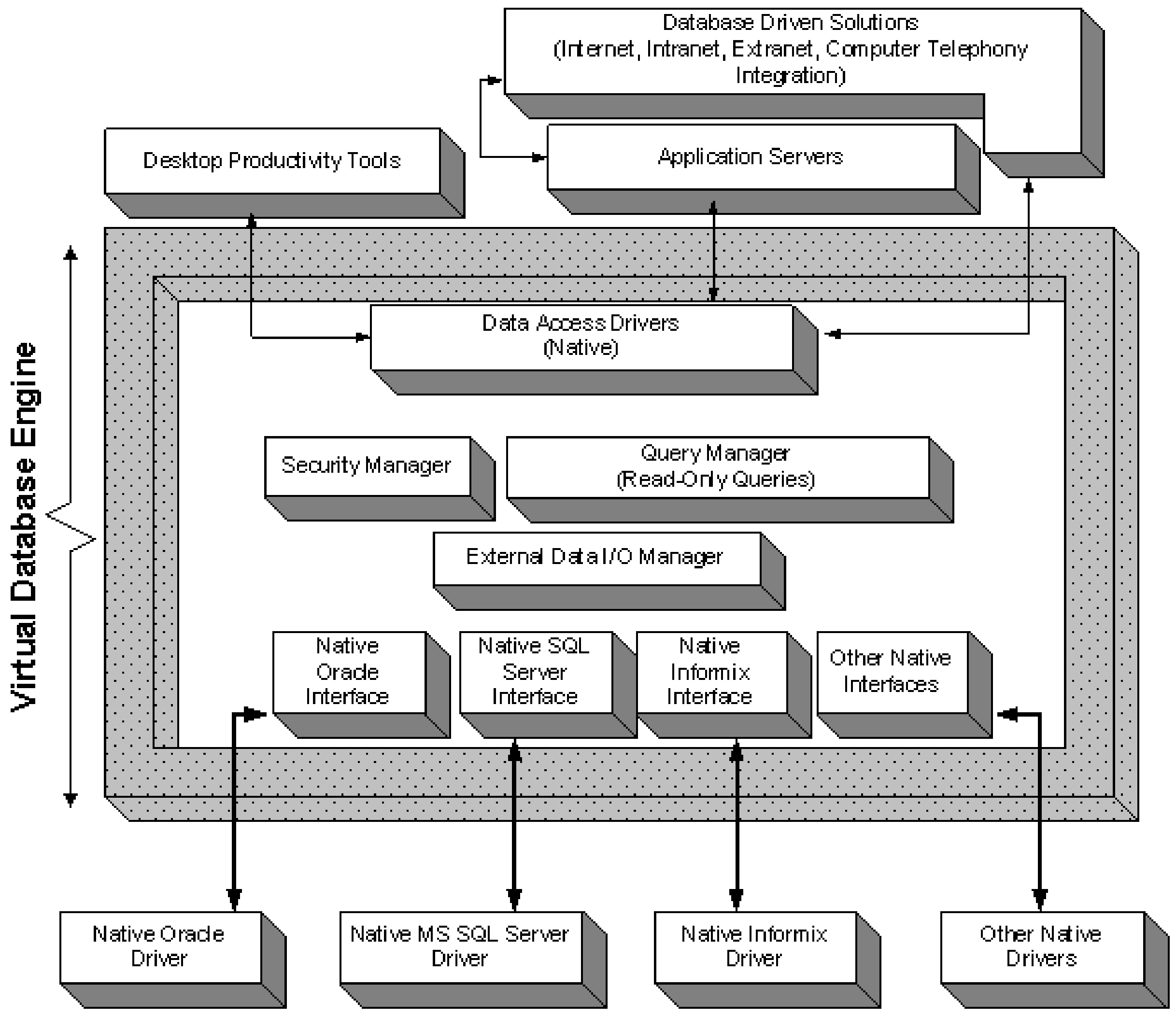
The connection that a socket provides can be connection-oriented or connectionless. *Connection-oriented* communication implies that a connection is established, and a dialog between the programs follows. The program that provides the service (the server program) establishes the available socket that is enabled to accept incoming connection requests. Optionally, the server can assign a name to the service that it supplies, which allows clients to identify where to obtain and how to connect to that service. The client of the service (the client program) must request the service of the server program. The client does this by connecting to the distinct name or to the attributes associated with the distinct name that the server program has designated. It is similar to dialing a telephone number (an identifier) and making a connection with another party that is offering a service (for example, a plumber). When the receiver of the call (the server, in this example, a plumber) answers the telephone, the connection is established. The plumber verifies that you have reached the correct party, and the connection remains active as long as both parties require it.

Connectionless communication implies that no connection is established, over which a dialog or data transfer can take place. Instead, the server program designates a name that identifies where to reach it (much like a post-office box). If you send a letter to a post office box, you cannot be absolutely sure that the receiver got the letter. You might need to wait for a response to your letter. There is no active, real-time connection, in which data is exchanged.

Virtual Database Engines Defined

. A **Virtual Database** (**VDB**) **Engine** is a UDA middleware format that transparently brings local and or remote heterogeneous **databases** together using logical **database** references called Data Source Names (DSNs).

A VDB **Engine's** data I/O occurs via low-level data access interfaces to underlying **database engines** or data sources. ... OLE-**DB** from Microsoft is also emerging as a new low-level data access standard for relational and non-relational data in the Microsoft Component Object Model (COM) world


Database Driven Solutions (Internet, Intranet, Extranet, Computer Telephony Integration)
Desktop Productivity Tools
Application Servers
Virtual Database Engine
Data Access Drivers (Native)
Security Manager
Query Manager (Read-Only Queries)
External Data I/O Manager
Native Oracle Interface
Native SQL Server Interface
Native Informix Interface
Other Native Interfaces
Native Oracle Driver
Native MS SQL Server Driver
Native Informix Driver
Other Native Drivers

Virtual Database Engines Defined

A Virtual Database (VDB) Engine is a UDA middleware format that transparently brings local and or remote heterogeneous databases together using logical database references called Data Source Names (DSNs). A VDB Engine exposes Metadata and Data held within these heterogeneous DSNs to clients applications and services homogeneously.

VDB Engines presume the existence of a number of Database Engines and Data Access Drivers provided by a variety of database vendors within an organization. VDB Engines provide transparent access to these heterogeneous databases via DSNs associated with the relevant data access drivers without exposing end-users or developers to the intricacies of heterogeneous data access.

Data Source Names (DSNs)

A Data Source Name is a logical reference that exposes database to standards compliant or native data access drivers. DSNs provide a flexible naming and binding service for database driven applications developers and end-users alike. Applications no longer need to be inextricably linked to specific database names or specific database engines.

The Need for Virtual Database Engines

Situation Analysis

As computer hardware, network protocols, database engines, applications, application servers, and desktop productivity tools, proliferate the enterprise, integration of disparate applications from disparate vendors is becoming an all too common problem.

Add the emergence of standards based Distributed Computing galvanized by the Internet infrastructure and associated Internet protocols to this picture, and the need for Integration is even higher.

Increasing the industry at large is looking to a new technology deliverable known as Universal Data Access Middleware to address these systems integration pains.

"With Universal Data Access (UDA), customers receive all of the benefits of a high-level and consistent Application Programming Interface (API) that abstracts all the database complexities while providing a capability that can be specified, controlled, and managed on its own to optimize the near universal need of programs for data access".

First Generation Virtual Database Products

Although the strict VDB definition may be new, there are a number of products that have been around for a while that attempt to address VDB issues. The list of such products includes The Microsoft JET Engine, Borland Database Engine (BDE), and IBM DataJoiner.

Microsoft JET

The Microsoft JET Engine lies at the heart of Microsoft Access, it is the piece of technology that allows you to link external and typically remote database tables into your local Access space via ODBC Data Sources. Once this link process has been completed, Access allows you to build Queries, Reports, Forms etc. using these external database tables as though they were Local Access tables. JET can also link to external tables hosted within desktop database engines via native interfaces.

The Microsoft JET Engine services are exposed via Microsoft provided data access interfaces such as: DAO, ADO, and OLE-DB. These interfaces are integral parts of most Microsoft applications, thereby exposing the benefits of the JET VDB transparently.

Borland Database Engine

The Borland Database Engine (BDE) from Inprise like the Microsoft JET Engine also facilitates external table linkage via ODBC Data Sources. The BDE also lets you link to external database tables via native database interfaces and there is no restriction to desktop database engines when you adopt this approach.

Although the BDE has a published set of APIs, it is predominantly used by Inprise applications in very much the same way JET is used by Microsoft applications.

IBM DataJoiner

DataJoiner from IBM provides the ability access heterogeneous data sources via IBM DB/2 Client Application Enablers. It does support ODBC and JDBC as client interfaces and makes use of Native or ODBC based data access for external Data I/O.

VDB Implementation Issues

The essential components that affect the implementation of VDB Engines are, High-Level Data Access Interfaces, Low-Level Data Access Interfaces and Traditional Database Functionality.

High-Level Data Access Interfaces

A VDB Engine’s capabilities are exposed via High Level Data Access interfaces. For the purpose of this document, a high level data access interface is an interface utilized predominantly by applications, as opposed to middleware developers for achieving application database independence. A high level data access interface sits atop Low-Level data access interfaces, providing an abstraction layer that serves to simplifying the process of database independent application development.

A number of High Level Data Access standards exist today, the more prevalent being: Data Access Objects (DAO), Remote Data Objects (RDO), ActiveX Data Objects (ADO), OLE-DB, JavaBlend, InfoBus.

It is important to note that low-level Data access interfaces such as ODBC, UDBC, JDBC and OLE-DB transparently serve the high-level interfaces mentioned in the section above. Thus, in most cases VDB vendors will treat ODBC, UDBC, JDBC, and OLE-DB as high-level interfaces by providing VDB data access drivers conforming to these standards as part of the VDB deliverable.

Low-Level Data Access Interfaces

A VDB Engine’s data I/O occurs via low-level data access interfaces to underlying database engines or data sources. In recent times the Open Database Connectivity (ODBC) API and the X/Open SQL Call Level Interface (CLI) have emerged as the dominant industry wide Low-Level Data Access Standards. OLE-DB from Microsoft is also emerging as a new low-level data access standard for relational and non-relational data in the Microsoft Component Object Model (COM) world. While JDBC is emerging like wise as the low-level data access standard for the burgeoning Java world.

A VDB may also be a Native Database Interface Client, making use of database engine vendor provided data access interfaces. Native interfaces are based upon Embedded SQL, an older format Low-Level data access interface that preceded the X/Open SQL CLI. It is important to note that ODBC from Microsoft, JDBC from JavaSoft, and UDBC from OpenLink Software are all derived from the X/Open SQL CLI.

Traditional Database Functionality

The degree to which a VDB implements a traditional database engine's functionality has a direct bearing on the intrinsic value of a VDB engine. Traditional database functionality is extensive, but for the purposes of this document, a core set of functionality common to all commercial database engines has been assembled. The functionality list includes:

Query Language Support - standard syntax for interrogating, manipulating, describing, and securing data contained within a database. Examples include the Structured Query Language (SQL) for relational databases and the Object query Language (OQL) for Object and Object-Relational Databases.

Query Processor – the mechanism used by a database engine to convert Query Language Statements into actual data retrieval instructions. In addition, this database component is responsible for ensuring Query Language syntax conformance, Query Execution Plan Assembly and Query Fulfillment.

Standard Data Types Support – data contained within a database must be describable using standard data types e.g. Character, Number, Date, etc.

VIEW Support – pre constructed query statements stored within a database, for the purpose of query simplification, or content and structural security.

Stored Procedure Support – Stored Procedures facilitate the embedding of application programming logic within a database. Their pre-compiled nature enhances data access performance by reducing message hops between database servers and database clients.

Scrollable Cursor Support – the process by which the result of a database query (known as a result-set) is traversed. Traversal occurs in either direction, backwards or forwards, using result-set chunks (known as row-sets). Resultset scrolling occurs when database engines exchange data with database clients.

Concurrency Control – the process through which a database engine supports multiple sessions running concurrently, across multiple database users and database client applications without compromising underlying data integrity or introducing quantum increases in application response times.

Transaction Support - ensures that database instructions can be grouped into logical units of execution that are Atomic, Consistent, Isolated from the effect of other units of execution affecting the same underlying data, and Durable.

Transaction Isolation - describes the ability of a database engine to provide transaction process partitioning options called Isolation Levels, that offer different ways of managing the effects of multiple and concurrent transactions affecting the same underlying data.

Distributed Transaction Support – describes the ability to preserve transaction atomicity, consistency, integrity, and durability across database servers hosted on the same or different database server machines within a networked environment. This involves supporting transaction Commits and Rollbacks using a 2-phase commit protocol.

User Definable Type Support – this is how a database engine allows end-users extend its base functionality. This is achieved by providing interfaces that allow end-users create new ways in which a database engine's data is described and manipulated.

Federated Database Support –data access and manipulation across database servers resident on the same machine.

Distributed Database Support - data access, and manipulation across database servers resident on the different machines within a networked environment.

Security – the process by which data, and data transmission is protected using a combination of database and operating system privileges, roles and roles hierarchies. It also includes the ability of a database engine to protect data transmitted to its clients using data encryption.

Virtual Database Engine Components

The prior section outlined the critical implementation issues that affect the development and implementation of VDB Engines. These issues form the basis around which a component based framework for depicting VDB architectures has been derived.

The components that comprise a VDB Engine framework are as follows:

Data Access Drivers

The VDB component that forms the entry point to the VDB Engine's services, these drivers may or may not conform to industry standards. Applications and Services that sit atop a VDB Engine must have their data access layers written to the same

Application Programming Interfaces (APIs) implemented by the Data Access Drivers provided by a VDB engine.

Security Manager

The VDB component that is responsible for protecting data and data transmission (using encryption) within the VDB Engine’s domain. It is also responsible for managing Application, User, Group, Role and Domain privileges as they relate to the creation, manipulation and destruction of VDB data and metadata.

Query Manager

The VDB component that handles queries presented to it by the VDB Engine’s data access drivers. It provides query syntax checking, query execution plan compilation, and query fulfillment services. A query processor is built in conformance to one or more query language specifications, the most notable being the Structured Query Language (SQL) for relational database engines, and the Object Query Language (OQL) for Object-Relational and Object Database engines.

Meta Data Manager

The VDB component that provides the Query Processor with information about the data entities from which the Query Processor’s execution plan is derived. Metadata managers are also the components responsible for linking external data sources into the VDB domain and directing the Query Processor to the appropriate Data I/O manager.

Transaction Manager

The Transaction Manager component ensures that transactions are Atomic (clearly distinguishable units), Consistent (thereby preserving integrity of data), Isolated from the effect of other transactions, and Durable (such that the effects of committed transactions survive failure). The Transaction Manager ensures VDB Engines are capable of supporting Online Transaction Processing (OLTP) and Distributed Transaction oriented applications and services. Transaction Managers may be standards based implementing X/Open’s XA Resource Manager Specifications. Distributed transaction support is implemented by using a two-phase commit protocol.

Concurrency Manager

The VDB component that ensures client applications and services are capable of opening multiple concurrent sessions that execute data INSERTS, UPDATES and DELETIONS, without implicitly reducing application response times or compromising data integrity. Concurrency control is delivered in one of two formats, Optimistic or Pessimistic depending on the response times desired by VDB client applications or services.

Local I/O Manager

VDB Engine’s that provide local data storage uses this component for reading and writing data to disk. This is how a VDB provides traditional database engine data storage services.

External Data I/O Manager

VDB component that handles data reads and writes to external data sources. The External Data I/O Manager be implemented using standard data access interfaces such as ODBC, JDBC, UDBC, OLE-DB or Native data source interfaces.

Replication Manager

Component that manages data migration and synchronization across two or more VDB servers within a distributed computing environment. This component acts as a data coordinator between the activities of Local Data I/O and External Data I/O Managers across VDB servers. The Replication Manager enables a VDB Engine offer automated bi-directional data, and metadata transformation services across heterogeneous data sources without end-user or developer intervention**.**

Web based middleware

Web services middleware consists of auxiliary products that work at the margins of a primary Web services application or facilitate the functionality between an application and an operating system. Web services middleware is also known as Web services management

Definition - *Web Services Middleware*

Web services middleware consists of auxiliary products that work at the margins of a primary Web services application or facilitate the functionality between an application and an operating system.

Web services middleware is also known as Web services management.

Explanation *Web Services Middleware*

Web services middleware plays the "middleman" role in the overall software architecture. The middleware layer can address things like security or cross-platform communications. It may also allow messaging between different software components. It is like a connective glue for a greater software structure.

Web services middleware works like a client-server architecture where the Web services application is the client and the middleware is the server, that is, it provides services to the client. This is how many engineers and developers think about the process when they add Web services middleware to a particular application.

Microsoft Jet Database Engine

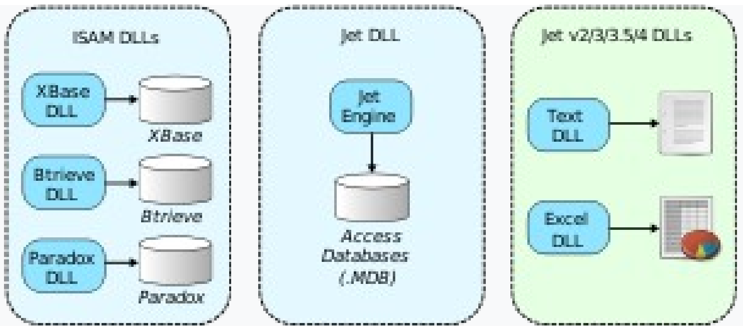
The **Microsoft Jet Database Engine** (also *Microsoft JET Engine* or simply *Jet*) is a database engine on which several Microsoft products have been built. The first version of Jet was developed in 1992, consisting of three modules which could be used to manipulate a database.

JET stands for *Joint Engine Technology*. Microsoft Access and Visual Basic use or have used Jet as their underlying database engine. However, it has been superseded for general use, first by Microsoft Desktop Engine (MSDE), then later by SQL Server Express. For larger database needs, Jet databases can be upgraded (or, in Microsoft parlance, "up-sized") to Microsoft's flagship SQL Server database product.

However, this does not mean that a MS Jet (Red) database cannot match MS SQL Server in storage capacity. A five billion record MS Jet (Red) database with compression and encryption turned on requires about one terabyte of disk storage space, being composed of hundreds of (*.mdb) files, each acting as partial table, and not as a database in itself.

Architecture

Jet, being part of a relational database management system (RDBMS), allows the manipulation of relational databases. It offers a single interface that other software can use to access Microsoft databases and provides support for security, referential integrity, transaction processing, indexing, record and page locking, and data replication. In later versions, the engine has been extended to run SQL queries, store character data in Unicode format, create database views and allow bi-directional replication with Microsoft SQL Server.



There are three modules to Jet: One is the *Native Jet ISAM Driver*, a dynamic link library (DLL) that can directly manipulate Microsoft Access database files (MDB) using a (random access) file system API. Another one of the modules contains the *ISAM Drivers*, DLLs that allow access to a variety of Indexed Sequential Access Method ISAM databases, among them xBase, Paradox, Btrieve and FoxPro, depending on the version of Jet. The final module is the *Data Access Objects* (DAO) DLL. DAO provides an API that allows programmers to access JET databases using any programming language

Locking

Jet allows multiple users to access the database concurrently. To prevent that data from being corrupted or invalidated when multiple users try to edit the same record or page of the database, Jet employs a locking policy. Any single user can modify only those database records (that is, items in the database) to which the user has applied a lock, which gives exclusive access to the record until the lock is released. In Jet versions before version 4, a page locking model is used, and in Jet 4, a record locking model is employed. Microsoft databases are organized into data "pages", which are fixed-length (2 kB before Jet 4, 4 kB in Jet 4) data structures. Data is stored in "records" of variable length that may take up less or more than one page. The page locking model works by locking the pages, instead of individual records, which though less resource-intensive also means that when a user locks one record, all other records on the same page are collaterally locked. As a result, no other user can access the collaterally locked records, even though no user is accessing them and there is no need for them to be locked. In Jet 4, the record locking model eliminates collateral locks, so that every record that is not in use is available.

There are two mechanisms that Microsoft uses for locking: *pessimistic locking*, and *optimistic locking*. With pessimistic locking, the record or page is locked immediately when the lock is requested, while with optimistic locking, the locking is delayed until the edited record is saved. Conflicts are less likely to occur with optimistic locking, since the record is locked only for a short period of time. However, with optimistic locking one cannot be certain that the update will succeed because another user could lock the record first. With pessimistic locking, the update is guaranteed to succeed once the lock is obtained. Other users must wait until the lock is released in order to make their changes. Lock conflicts, which either require the user to wait, or cause the request to fail (usually after a timeout) are more common with pessimistic locking.

Transaction processing

Jet supports transaction processing for database systems that have this capability. (ODBC systems have one-level transaction processing, while several ISAM systems like Paradox do not support transaction processing.) A transaction is a series of operations performed on a database that must be done together — this is known as atomicity and is one of the ACID (Atomicity, Consistency, Isolation, and Durability), concepts considered to be the key transaction processing features of a database management system. For transaction processing to work (until Jet 3.0), the programmer needed to begin the transaction manually, perform the operations needed to be performed in the transaction, and then commit (save) the transaction. Until the transaction is committed, changes are made only in memory and not actually written to disk.[1] Transactions have a number of advantages over independent database updates. One of the main advantages is that transactions can be abandoned if a problem occurs during the transaction. This is called rolling back the transaction, or just rollback, and it restores the state of the database records to precisely the state before the transaction began. Transactions also permit the state of the database to remain consistent if a system failure occurs in the middle of a sequence of updates required to be atomic. There is no chance that only some of the updates will end up written to the database; either all will succeed, or the changes will be discarded when the database system restarts. With ODBC's in-memory policy, transactions also allow for many updates to a record to occur entirely within memory, with only one expensive disk write at the end.

Implicit transactions were supported in Jet 3.0. These are transactions that are started automatically after the last transaction was committed to the database. Implicit transactions in Jet occurred when an SQL DML statement was issued. However, it was found that this had a negative performance impact in 32-bit Windows (Windows 95, Windows 98), so in Jet 3.5 Microsoft removed implicit transactions when SQL DML statements were made.

Data integrity

Jet enforces entity integrity and referential integrity. Jet will by default prevent any change to a record that breaks referential integrity, but Jet databases can instead use propagation constraints (cascading updates and cascading deletes) to maintain referential integrity.

Jet also supports "business rules" (also known as "constraints"), or rules that apply to any column to enforce what data might be placed into the table or column. For example, a rule might be applied that does not allow a date to be entered into a date_logged column that is earlier than the current date and time, or a rule might be applied that forces people to enter a positive value into a numeric only field.

Security

Access to Jet databases is done on a per user-level. The user information is kept in a separate system database, and access is controlled on each object in the system (for instance by table or by query). In Jet 4, Microsoft implemented functionality that allows database administrators to set security via the SQL commands CREATE, ADD, ALTER, DROP USER and DROP GROUP. These commands are a subset of ANSI SQL 92 standard, and they also apply to the GRANT/REVOKE commands.[1] When Jet 2 was released, security could also be set programmatically through DAO.

Queries

Queries are the mechanisms that Jet uses to retrieve data from the database. They can be defined in Microsoft QBE (Query By Example), through the Microsoft Access SQL Window or through Access Basic's Data Access Objects (DAO) language. These are then converted to an SQL SELECT statement. The query is then compiled — this involves parsing the query (involves syntax checking and determining the columns to query in the database table), then converted into an internal Jet query object format, which is then tokenized and organised into a tree like structure. In Jet 3.0 onwards these are then optimised using the Microsoft Rushmore query optimisation technology. The query is then executed and the results passed back to the application or user who requested the data.

Jet passes the data retrieved for the query in a dynaset. This is a set of data that is dynamically linked back to the database. Instead of having the query result stored in a temporary table, where the data cannot be updated directly by the user, the dynaset allows the user to view and update the data contained in the dynaset. Thus, if a university lecturer queries all students who received a distinction in their assignment and finds an error in that student's record, they would only need to update the data in the dynaset, which would automatically update the student's database record without the need for them to send a specific update query after storing the query results in a temporary table.

Unit -7

Database Recovery Techniques

Database systems, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database. There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur in the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read_item(T, X): This log entry records that transaction T reads the value of database item X.
- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- abort(T): This records that transaction T has been aborted.
- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process

- **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to old-value.There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
- **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**
- **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point.

However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**

- **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.
- **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to original are updated to refer new replacement page.

Some of the backup techniques are as follows :

- **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.
- **Differential backup –** It stores only the data changes that have occurred since last full database backup. When same data has changed many times since last full database backup, a differential backup stores the most recent version of changed data. For this first, we need to restore a full database backup.
- **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transaction that had happened to the database. Through this, the database can be recovered to a specific point in time. It is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost.

<u>What is database security</u>

Database security encompasses a range of security controls designed to protect the Database Management System (DBMS). The types of database security measures your business should use include protecting the underlying infrastructure that houses the database such as the network and servers), securely configuring the DBMS, and the access to the data itself.

Database security controls

Database security encompasses multiple controls, including system hardening, access, DBMS configuration, and security monitoring. These different security controls help to manage the circumventing of security protocols.

System hardening and monitoring

The underlying architecture provides additional access to the DBMS. It is vital that all systems are patched consistently, hardened using known security configuration standards, and monitored for access, including insider threats.

DBMS configuration

It is critical that the DBMS be properly configured and hardened to take advantage of security features and limit privileged access that may cause a misconfiguration of expected security settings. Monitoring the DBMS configuration and ensuring proper change control processes helps ensure that the configuration stays consistent.

Authentication

Database security measures include authentication, the process of verifying if a user’s credentials match those stored in your database, and permitting only authenticated users access to your data, networks, and database platform.

Access

A primary outcome of database security is the effective limitation of access to your data. Access controls authenticate legitimate users and applications, limiting what they can access in your database. Access includes designing and granting appropriate user attributes and roles and limiting administrative privileges.

Database auditing

Monitoring (or auditing) actions as part of a database security protocol delivers centralized oversight of your database. Auditing helps to detect, deter, and reduce the overall impact of unauthorized access to your DBMS.

Backups

A data backup, as part of your database security protocol, makes a copy of your data and stores it on a separate system. This backup allows you to recover lost data that may result from hardware failures, data corruption, theft, hacking, or natural disasters.

Encryption

Database security can include the secure management of encryption keys, protection of the encryption system, management of a secure, off-site encryption backup, and access restriction protocols.

Application security

Database and application security framework measures can help protect against common known attacker exploits that can circumvent access controls, including SQL injection.

Why is database security important

Safeguarding the data your company collects and manages is of utmost importance. Database security can guard against a compromise of your database, which can lead to financial loss, reputation damage, consumer confidence disintegration, brand erosion, and non-compliance of government and industry regulation.
Database security safeguards defend against a myriad of security threats and can help protect your enterprise from:

- Deployment failure
- Excessive privileges
- Privilege abuse
- Platform vulnerabilities
- Unmanaged sensitive data
- Backup data exposure
- Weak authentication
- Database injection attacks

Data Mining

In general terms, **“Mining”** is the process of extraction of some valuable material from the earth e.g. coal mining, diamond mining etc. In the context of computer science, **“Data Mining”** refers to the extraction of useful information from a bulk of data or data warehouses. One can see that the term itself is a little bit confusing. In case of coal or diamond mining, the result of extraction process is coal or diamond. But in case of Data Mining, the result of extraction process is not data!! Instead, the result of data mining is the patterns and knowledge that we gain at the end of the extraction process. In that sense, Data Mining is also known as Knowledge Discovery or Knowledge Extraction.

Gregory Piatetsky-Shapiro coined the term “Knowledge Discovery in Databases” in 1989. However, the term ‘data mining’ became more popular in the business and press communities. Currently, Data Mining and Knowledge Discovery are used interchangeably.

Now a days, data mining is used in almost all the places where a large amount of data is stored and processed. For example, banks typically use ‘data mining’ to find out their prospective customers who could be interested in credit cards, personal loans or insurances as well. Since banks have the transaction details and detailed profiles of their customers, they analyze all this data and try to find out patterns which help them predict that certain customers could be interested in personal loans etc.

Main Purpose of Data Mining

Basically, the information gathered from Data Mining helps to predict hidden patterns, future trends and behaviors and allowing businesses to take decisions.

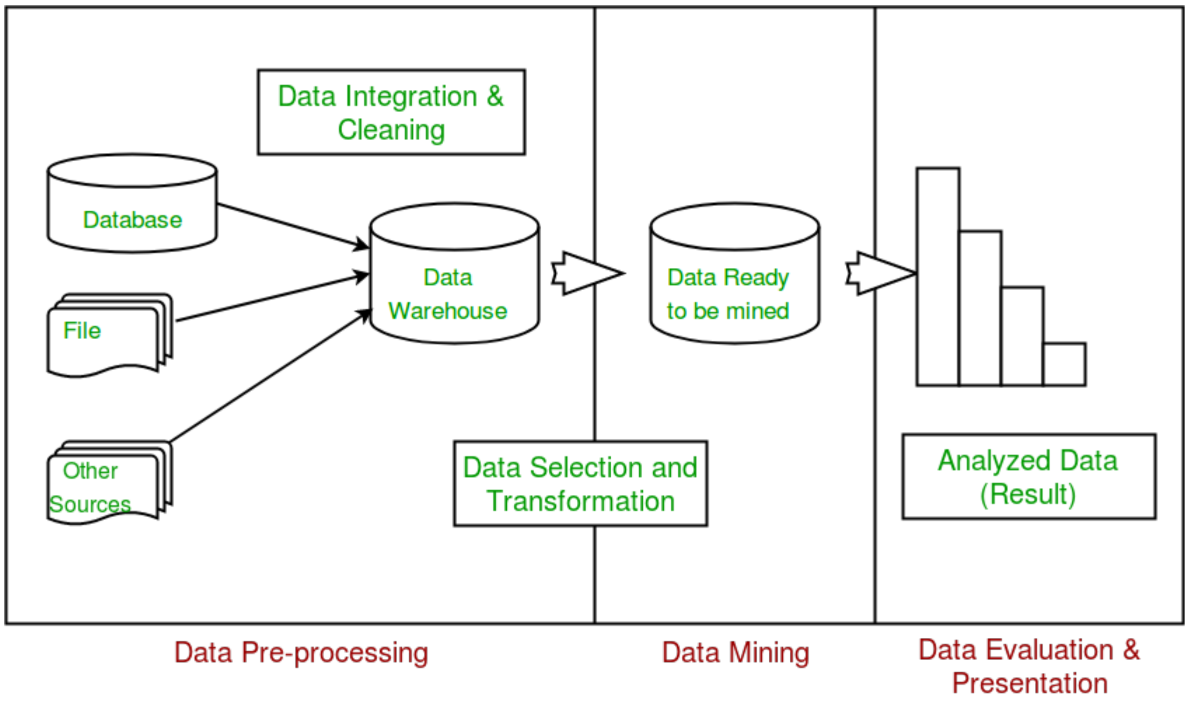
Technically, data mining is the computational process of analyzing data from different perspective, dimensions, angles and categorizing/summarizing it into meaningful information.

Data Mining can be applied to any type of data e.g. Data Warehouses, Transactional Databases, Relational Databases, Multimedia Databases, Spatial Databases, Time-series Databases, World Wide Web.

Data Mining as a whole process

The whole process of Data Mining comprises of three main phases:

1. Data Pre-processing – Data cleaning, integration, selection and transformation takes place
2. Data Extraction – Occurrence of exact data mining
3. Data Evaluation and Presentation – Analyzing and presenting results



In future articles, we will cover the details of each of these phase.

Applications of Data Mining

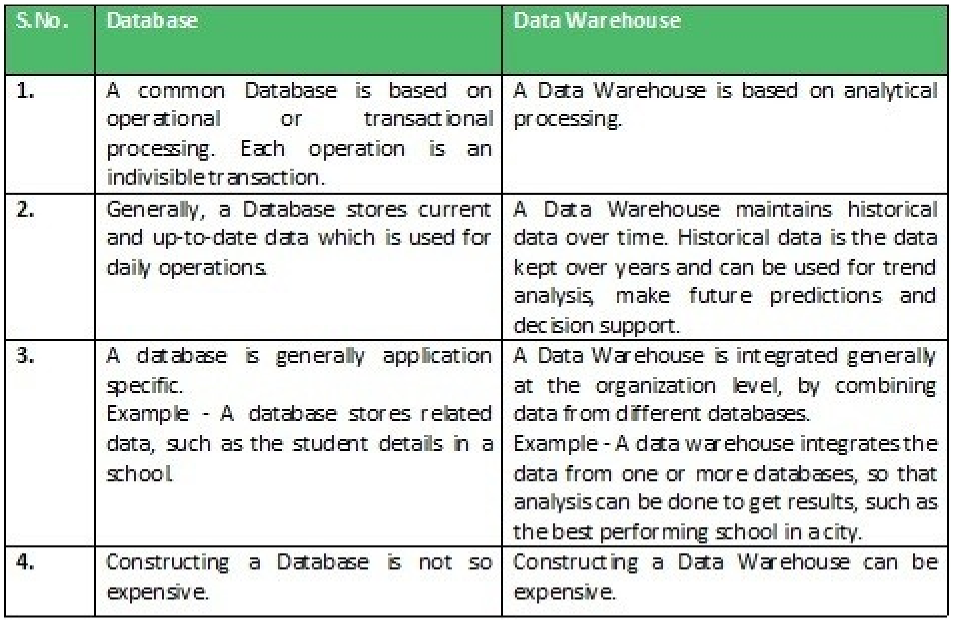
1. Financial Analysis
2. Biological Analysis
3. Scientific Analysis
4. Intrusion Detection

5. Fraud Detection
6. Research Analysis
Real life example of Data Mining – Market Basket Analysis
Market Basket Analysis is a technique which gives the careful study of purchases done by a customer in a super market. The concept is basically applied to identify the items that are bought together by a customer. Say, if a person buys bread, what are the chances that he/she will also purchase butter. This analysis helps in promoting offers and deals by the companies. The same is done with the help of data mining.

Data Warehousing

Background
A Database Management System (DBMS) stores data in the form of tables, uses ER model and the goal is ACID properties. For example a DBMS of college has tables for students, faculty, etc.
A **Data Warehouse** is separate from DBMS, it stores huge amount of data, which is typically collected from multiple heterogeneous source like files, DBMS, etc. The goal is to produce statistical results that may help in decision makings. For example, a college might want to see quick different results, like how is the placement of CS students has improved over last 10 years, in terms of salaries, counts, etc.
Need of Data Warehouse
An ordinary Database can store MBs to GBs of data and that too for a specific purpose. For storing data of TB size, the storage shifted to Data Warehouse. Besides this, a transactional database doesn’t offer itself to analytics. To effectively perform analytics, an organization keeps a central Data Warehouse to closely study its business by organizing, understanding and using its historic data for taking strategic decisions and analyzing trends.

Data Warehouse vs DBMS



S.No.	Database	Data Warehouse
1.	A common Database is based on operational or transactional processing. Each operation is an indivisible transaction.	A Data Warehouse is based on analytical processing.
2.	Generally, a Database stores current and up-to-date data which is used for daily operations.	A Data Warehouse maintains historical data over time. Historical data is the data kept over years and can be used for trend analysis, make future predictions and decision support.
3.	A database is generally application specific. Example - A database stores related data, such as the student details in a school.	A Data Warehouse is integrated generally at the organization level, by combining data from different databases. Example - A data warehouse integrates the data from one or more databases, so that analysis can be done to get results, such as the best performing school in a city.
4.	Constructing a Database is not so expensive.	Constructing a Data Warehouse can be expensive.

Example Applications of Data Warehousing
Data Warehousing can be applicable anywhere where we have huge amount of data and we want to see statistical results that help in decision making.

- **Social Media Websites:** The social networking websites like Facebook, Twitter, Linkedin etc. are based on analyzing large data sets. These sites gather data related to members, groups, locations etc. and store it in a single central repository. Being large amount of data, Data Warehouse is needed for implementing the same.
- **Banking :** Most of the banks these days use warehouses to see spending patterns of account/card holders. They use this to provide them special offers, deals, etc.
- **Government :** Government uses data warehouse to store and analyze tax payment which is used to detect tax thefts.

There can be many more applications in different sectors like E-Commerce, Telecommunication, Transportation Services, Marketing and Distribution, Healthcare and Retail.