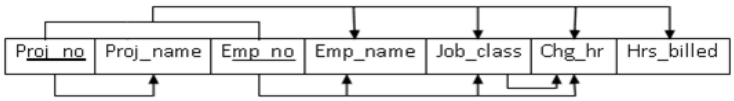
DESCRIPTIVE ANSWER OF DBMS

Q1 . Consider a dependency diagram of relation R and normalize it up to third normal form.



ANS .

- Normalized Relation,
- Employees (Proj_no, Emp_no, Proj_name, Emp_name, Job_Class, Chg_Hr, Hrs_Billed)
- With set of FDs

  Proj_no, Emp_no → Proj_name, Emp_name, Job_Class, Chg_Hr, Hrs_Billed

  Emp_no → Emp_name, Job_Class, Chg_Hr

  Proj_no → Proj_name

  Job_Class → Chg_Hr

Q2. Explain conflict and view serializability with suitable examples .

ANS .

Conflict serializability

The database system must control concurrent execution o transactions which ensure that the database state remains i consistent state.

Conflict

A pair of consecutive database actions (reads, writes) is i conflict if changing their order would change the result of at leas one of the transactions.

		Transaction T_j	
		Read(D)	Write(D)
Transaction T_i	Read(D)	No Conflict	Conflict
	Write(D)	Conflict	Conflict

Consider schedule S has two consecutive instructions I_i and I_j from transactions T_i and T_j respectively. If I_i and I_j access to different data items then they will not conflict and can be swapped, without any problem.

If I_i and I_j **access to same data item D** then consider following consequences :

I_i = READ (D), I_j = READ (D) then **no conflict** as they only read value.

This operation is called as non conflicting swap.

I_i = READ (D), I_j = WRITE (D) then they **conflict** and cannot be swapped.

I_i = WRITE (D), I_j = READ (D) then they **conflict** and cannot be swapped.

I_i = WRITE (D), I_j = WRITE (D) then they **conflict** and cannot be swapped.

So we can say that instructions conflict if both consecutive instructions operate on same data item and from different transactions and one of them is WRITE operation.

If I_i and I_j **access to different data item D** then consider following all consequences **no conflict** as they only read or writing different values.

I_i = READ (D)/WRITE (D), I_j = READ (P) / WRITE (P) then **no conflict** as they only reading or writing different data.

The following set of actions is conflicting :

T_1:R(X), T_2:W(X), T_3:W(X)

While the following sets of actions are not :

T_1:R(X), T_2:R(X), T_3:R(X)

T_1:R(X), T_2:W(Y), T_3:R(X)

Conflict equivalence

Two schedules are conflict equivalent if they can be turned into one another by a sequence of non conflicting swaps of adjacent actions.

Example

A schedule is conflict serializable if it is conflict equivalent to a serial schedule.

T_1	T_2
Read(P)	
Write(P)	
	Read(P)
Read(Q)	
	Write(P)
Write(Q)	
	Read(Q)
	Write(Q)

Fig. 10.4 : Schedule S

Instruction WRITE(P) of T_1 and READ(P) of T_2 cannot be swapped as they conflict.

Ans. : View Serializability

View equivalence is less strict than conflict equivalence, but it is like conflict equivalence based on only the read and write operations of transactions.

Conditions for view equivalence

Let, D = Data item
S_1, S_2 = Transaction schedules
T_i, T_j = Database transaction

Schedules S_1 and S_2 are view equivalent if they satisfy following conditions for each data item D.

(a) S_1 and S_2 must have same transactions included and also they are performing same operations on same data. If T_i reads initial value of D in S_1, then T_i also reads initial value of D in S_2.

(b) If T_i reads value of D written by T_j in S_1, then T_i also reads value of D written by T_j in S_2.

(c) If T_i writes final value of D in S_1, then T_i also writes final value of D in S_2.

First 2 conditions ensure that transaction reads same value in both schedules. Condition 3 ensures that final consistent state. If a concurrent schedule is view equivalent to a serial schedule of same transactions then it is **View serializable**.

Consider following schedule S_1 with concurrent transactions $<T_1, T_2, T_3>$. In both the schedules S_1 and a serial schedule $S_2<T_1, T_2, T_3>$ T_1 reads initial value of D. Transaction T_3 writes final value of D. So schedule S_1 satisfies all three conditions and is view serializable to $<T_1, T_2, T_3>$.

Example

Below two schedules S and R are view equivalent,

Schedule S		
T_1	T_2	T_3
Read(P)		
	Write (P)	
Write (P)		
		Write (P)

Schedule R		
T_1	T_2	T_3
Read(P)		
Write (P)		
	Write (P)	
		Write (P)

Q3. Explain deadlock handling in DBMS with suitable examples.

ANS.

1. A system is said to be in a state of deadlock if there exists a set of transactions such that every transaction in the set is waiting for another transaction to complete its execution.

Example :

Consider two transactions given below,

T_1 : This transaction first read and then write data on data item X, then after that performs read and write on data item Y.

T_1
Read (X)
Write (X)
Read (Y)
Write (Y)

T_2 : This transaction first read and then write data on data item Y, then after that performs read and write on data item X.

T_2
Read (Y)
Write (Y)
Read (X)
Write (X)

Consider above two transactions are executing using locking protocol as below,

T_1	T_2	
Lock-X(X)		
Read (X)		
Write (X)		
	Lock-X(Y)	

T_1	T_2	
	Read (Y)	
	Write (Y)	
Lock-X(Y)		Wait for transaction T_2 to Unlock Y
Read (Y)		
Write (Y)		
	Lock-X(X)	Wait for transaction T_1 to Unlock X
	Read (X)	
	Write (X)	

So in above schedule consider two transactions given below transaction T_1 is waiting for transaction T_2 to Unlock data item Y and transaction T_2 is waiting for transaction T_1 to Unlock data item X.

So system is in a deadlock state as there are set of transactions T_1 and T_2 such that, both are waiting for each other transaction to complete. This state is called as **Deadlock state.**

2. There are **two principle methods** to handle deadlock in system,

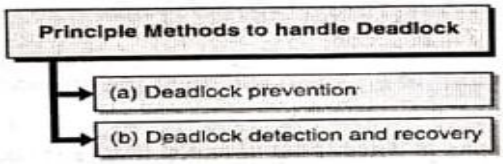


Fig. 12.7.1 : Principle Methods to handle Deadlock

(a) Deadlock prevention

We can use deadlock prevention techniques to ensure that the system will never enter a deadlock state.

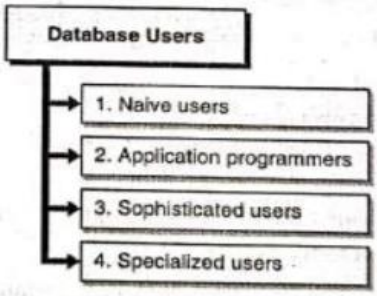
(b) Deadlock detection and recovery

We can allow the system to enter a deadlock state and then we can detect such state by **deadlock detection techniques** and try to recover from deadlock state by using and **deadlock recovery scheme.**

3. Both of above methods may result in transaction rollback.
4. Deadlock prevention is commonly used if the probability that the deadlock occurs in system is high else detection and recovery are more efficient if probability of deadlock occurrence is less.

Q4. What are different database users? Give responsibilities of DBA

ANS .



The various responsibilities of DBA are as follows,

Designing overall Database schema

The DBA is responsible for designing overall database schema (tables and fields). Al responsible for deciding on the data storage and access methods.

Selecting and installing database software and hardware.

The DBA selects the suitable DBMS software like Oracle, SQL Server or MySQL.

Designing Authorization/Access Control

The DBA will decide the user access levels and security checks for access and da manipulations.

Designing Recovery Procedures

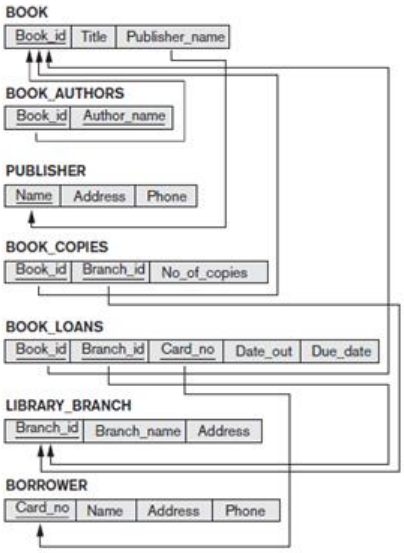
In order to take care of system crashes DBA needs to design the system recove procedures and also specifying techniques for monitoring database performance.

Operations Management

The operations management of database administration deals with data problems aris on a day-to-day basis. Specifically, the responsibilities include

- Investigation of errors found in the data.
- Supervision of restart and recovery procedures in the event of a failure.
- Supervision of reorganization of databases.
- Initiation and control of all periodic dumps of data.

Q5. Produce ER Diagram from the following relational database Schema.



ANS .

Q6. Book(book_id, title,author, cost)
Store(store_no, city, state, inventory_val)
Stock(store_no, book_id,quantity)

Consider above relational schema and formulate SQL queries for the following:

(i)Modify the cost of DBMS books by 10%
(ii)Find the author of the books which are available in Mumbai store
(iii)Find the title of the most expensive book
(iv)Find the total quantity of books in each store
(v) Add a new record in Book(Assume values as per requirement)

ANS.

Q7. Explain the transaction processing with the help of a state diagram?

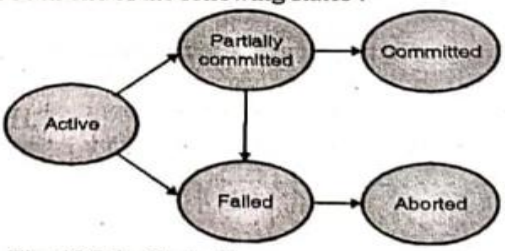
ANS.

(1) Introduction

- If transaction complete successfully it may be saved on to database server called as **committed** transaction.
- A committed transaction transfers database from one consistent state to other consistent state, which must persist even if there is a system failure.
- Transaction may not always complete its execution successfully. Such a transaction must be **aborted**.
- An aborted transaction must not have any change that the aborted transaction made to the database. Such changes must be undone or **rolled back**.
- Once a transaction is committed, we cannot undo its effects by aborting it.

(2) Transaction states

- A transaction must be in one of the following states :



(a) **Active**

- This is initial state of transaction execution.
- As soon as transaction execution starts it is in active state.
- Transaction remains in this state till transaction finishes.

(b) **Partially committed**

- As soon as last operation in transaction is executed transaction goes to partially committed state.
- At this condition the transaction has completed its execution and ready to commit on database server. But, it is still possible that it may be aborted, as the actual output may be there in main memory, and thus a hardware failure may prohibit its successful completion.

(c) **Failed**

A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution.

Example : In case of Hardware or logical errors occurs while execution.

(d) **Aborted**

- Failed transaction must be rolled back. Then, it enters the aborted state.
- Transaction has been rolled back restoring into prior state.
- In this stage system have two options :
  - **Restart the transaction :** A restarted transaction is considered to be a new transaction which may recover from possible failure.
  - **Kill the transaction :** Because the bad input or because the desired data were not present in the database an error can occurs. In this case we can kill transaction to recover from failure.

(e) **Committed**

- When last data item is written out, the transaction enters into the committed state.
- This state occurs after successful completion of transaction.
- A transaction is said to have terminated if has either committed or aborted.

Q8. Consider the schema R={A,B,C,D,E,F,G,H,I,J} and set of functional dependencies

F= {{A,B}→{C}, {A}→{D,E}, {B}→ {F}, {F}→{G,H},{D}→{I,J}}.

What is the key of R?

Decompose R into 2NF and 3NF relations.

ANS.

A→DE (given) => A→D and A→ D
Since A→D and D→IJ (given) => A→IJ
Using the union rule A→ ADEIJ, thus AB → ABDEIJ (augmentation)
Also AB→C (given) => AB→ ABCDEIJ.
Since B→F (given) and F→ GH (given), B→GH (transitivity)
Thus AB→AGH holds. Also AB→AF holds from B→F (given)
Finally, using the union rule AB→ABCDEFGHIJ.
So AB is a key. This can also be determined by calculating AB^+ with respect to the set F.

2NF	3NF	
R1 (A, B, C)	R1 (A, B, C)	
R2 (A,D,E,I,J)	R2.1 (A,D,E)	R2.2 (D, I, J)
R3 (B, F, G, H)	R3.1 (B, F)	R3.2 (F, G, H)

Q9. Explain log based recovery techniques with examples?

ANS .

(1) Introduction

- There can be problem in accessing database due to any reason causes a database system failure.
- The most widely used structure for recording database modifications is **Transaction log (or log).**
- The log is a sequence of log records, recording all the update activities done on the database by all database users.

(2) Transaction log

- Transaction log records are maintained to record various events during transaction processing.
- Transaction log file is recorded when transaction performs a write operation to record data changes.
- Log records to be useful for recovery from system and disk failures, the log must reside in stable storage.
- We assume that every log record is written to the end of the operation on stable storage as soon as log is created.
- Transactional log contains following data :

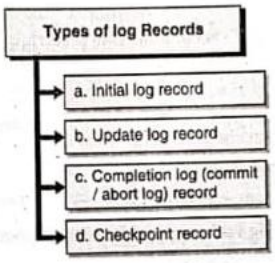
(i) **Transaction Identifier (T_i) :** This is the unique identifier of the transaction that is recorded at write operation.

(ii) **Data item Identifier(X) :** unique identifier to recognize data item written.

(iii) **Old Value (V_1) :** Value of the old data item.

(iv) **New Value (V_2) :** Value of new data item after the write is done.

- There are several types of log records.



(a) Initial log record

- To start a transaction initial transaction log is recorded.
- This log indicates that recording log file is started.
- Log Record :< T_n Start>
- Transaction (T_n) has started.

Example :

<T_1 Start> : Transaction T_1 is started.

(b) Update log record

- An update log record describes a single database write.
- It also includes the value of the bytes of page before and after the page change.
- **Log record :**< T_n, X, V_1, V_2>
- Transaction (T_n) has performed a write on data item X.
- It modify current value V_1 of data item X to updated value V_2 after the write.

Example :

<T_1 , A, 100,500> : Transaction T_1 changed value of A to 500.

Or <T_1 , A, 500> : Transaction T_1 changed value of A to 500.

(c) Completion log (commit / abort log) record

- If transaction completes operations successfully then commit a transaction or if any problem while executing transaction decision to abort and hence rollback a transaction.
- **Operational Update Log :** < T_n Commit > or < T_n Abort >
- It commits or rolls back the operations performed by transaction.

Example :

<T_1 Commit> : Transaction T_1 is committed to server.

Or <T_1 Rollback> : Transaction T_1 abort its operations.

(d) Checkpoint record

- Records a point when checkpoint has been made.
- These are used to speed up recovery.
- It also record information that eliminates the need to read a log's past.
- The time of recording varies according to checkpoint algorithm.
- **Log record :** < T_n Checkpoint A >
- It marks transaction status.

Example :

<T_1 Checkpoint A>: Transaction T_1 is committed to server.

(3) Recovery actions

Redo operation

- If a transaction has recorded commit operation before failure occurs, then transaction must be redone.
- Recovery Action : **REDO** (T_i)

Example :

REDO (T_1)

Undo operation

- If a transaction has not recorded commit operation before failure occurs, then transaction must be undone.
- Recovery Action : **UNDO** (T_i)

Example : UNDO (T_1)

Q10. Explain different types of Database users and the responsibilities of the DBA?

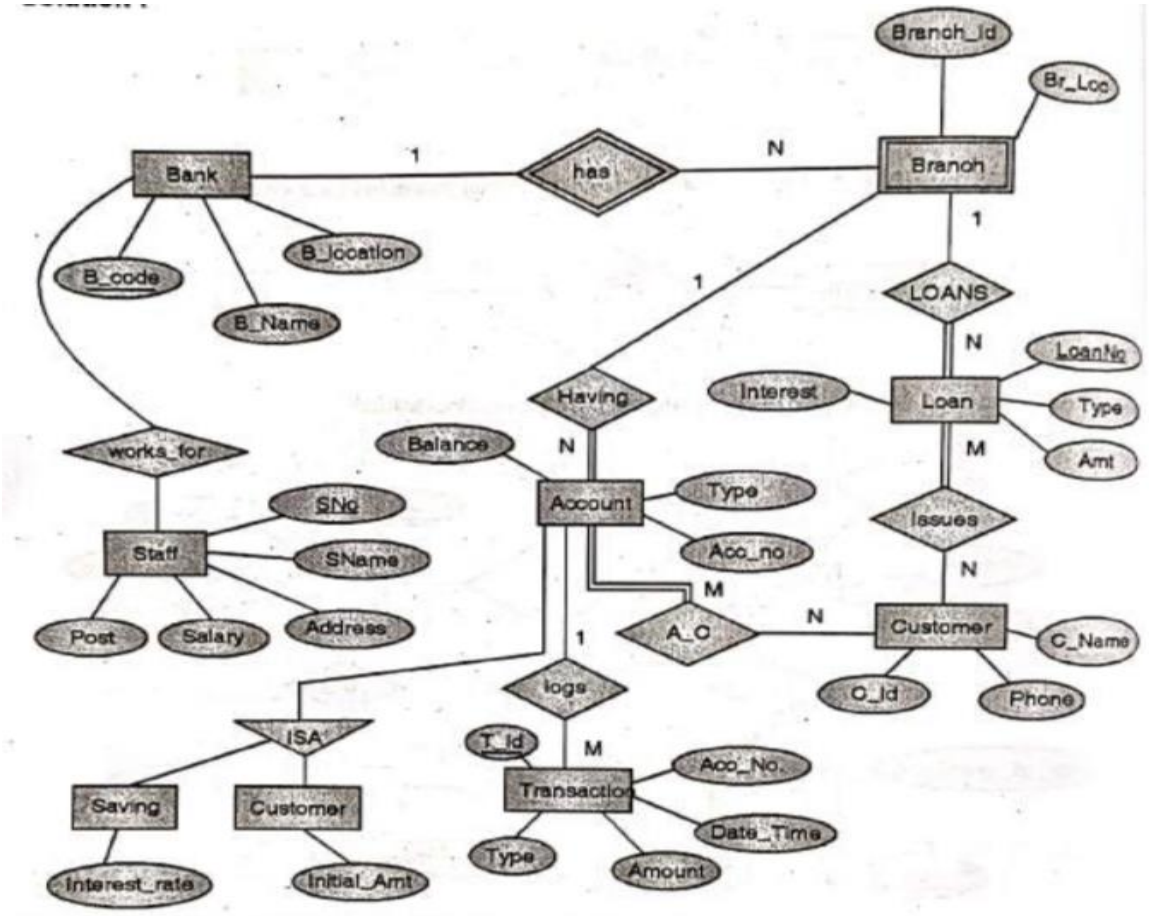
ANS . SAME ANSWER OF Q4

Q11. Design an EER schema for a **BANK** database.

Each bank can have multiple branches, and each branch can have multiple accounts and loans. Bank keeps the track of different types of Accounts (Saving_aacount, Checking_account) , Loans(Car_loans,Home_loans,…) , each account's Transaction (deposit, withdrawal,check,..) and each loan's Payments; both of these include the amount, date and time.

State any assumptions you make about the additional requirement clearly.

ANS .



Q12. Write SQL queries for the given database :

Emp(Eid, Ename, Sal, City)

Works(Eid, Cid)

Company(Cid, Cname, City)

i. Find the lowest paid employee.

ii. Find how many employees are working for the company 'ANZ Cooperation".

iii. Modify the database so that Joe now lives in “New York”.

iv. Find the total number of employees of each company.

v. Give all employees of ‘XYZ ‘company a 10% raise in salary.

ANS.

Q13. Explain the three levels of abstraction in DBMS including physical and logical data independence.

ANS .

Ans. : Database architecture

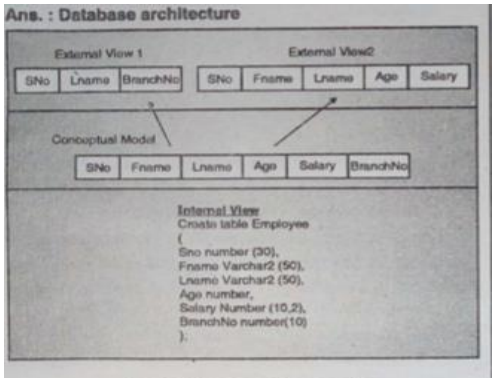


Fig. 2.1 : Database schema levels

(I) Internal Level (Physical Level)

The internal level is very close to physical storage of data. This level describes the physical storage structure of the data in database. The internal (or physical) database is stored on secondary storage devices, mainly the magnetic disk. Describes the complete details of data storage and various available access methods for the database.

At its ground level, it is stored in the form of bits with the physical addresses on the secondary storage device. At its highest level, it can be viewed in the form of files and simple data structures.

Internal view/ schema

The internal view defines the various stored data types and specifies what type of indexes exist, how stored fields are represented and so on. The internal schema uses a physical data model.

(II) Conceptual level

This level describes the structure of the whole database for a group of users. The conceptual model is also called as the data model or we can say data model is used to describe the conceptual schema when a database system is implemented.

The conceptual schema hides the internal details of physical storage and targets on describing entities, data types, relationships and constraints. The conceptual schema contains all the information to build relevant external records. As the conceptual model is derived from the physical model.

Conceptual view / schema

The conceptual view is a representation of the entire content of the database. The conceptual view includes definitions of each of the various conceptual data types.

(III) External level (view level)

The external level is the one closest to the user, i.e., it is related with the way data is viewed by individual end users. The external level includes a number of user views or external schemas.

Each external schema describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.

External views are the proper interface between the user and the database, as an individual user can hardly be expected to be interested in the entire database. The external model is derived from the conceptual model.

External view / schema

External schema consists of definitions of each of the various external data types in that external view.

(IV) Mapping

The processes of transforming requests and results between various levels of architecture are called mappings. These mappings may be time-consuming, so small databases do not support external views.

External / conceptual mapping : The DBMS must transform a request on an external schema into a request against the conceptual schema.

Conceptual / internal mapping : A certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

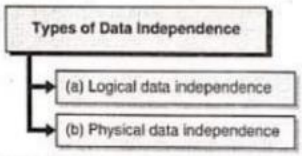


Fig. 2.2.1 : Types of Data Independence

(a) Logical data independence

- Logical data independence is a capacity to change the conceptual schema without havi any changes to external schemas. (or application programs)
- Separating the external views from the conceptual view enables us to change th conceptual view without affecting the external views. This separation is sometimes call logical data independence.

Example :

- We may change the conceptual schema by removing a data item. In this case the extern schemas that refer only to the remaining data should not be affected.

(b) Physical data independence

- Physical data independence is a capacity to change the internal schema without having any changes to conceptual schema.
- The separation of the conceptual view from the internal view enables us to provide a logical description of the database without the need to specify physical structures. This is often called physical data independence.

Example :

By creating additional access paths to improve the performance of retrieval. If the same data as before remains in the database, we should not have to change the conceptual schema.

Q14. Consider the given schema:

- **Employees (**Empid, Fname, Lname, Email, Phoneno, Hiredate, Jobid, Salary, Mid, Did**)**
- **Departments** (Did, Dname,Managerid)
- **Locations** (Did, City,State)

Write the SQL queries for the following:

1. List the employees who have a manager who works for a department based in Mumbai.
2. Give a 10% hike to all the Employees working in '**D01'** department.
3. Display the information of the employees whose first name starts with 'R' in descending order of their salary.
4. Find name of the department which are having more than 20 employees
5. Add a new record in departments(Assume values as per requirement)

ANS .

Q16. Explain 3NF .Consider relation r1 with the functional dependencies that hold on it.

r1(p, q, r, s, t)
p → q,r,s,t
s → t
check whether r1 is in 3NF or not .If it is not in 3NF decompose into 3NF.

ANS .

3NF

This normal form is given by E.F. Codd in 1971. This normal form introduced to minimize the transitive redundancy. To remove the data anomalies left in relational schema even after applying second normal form like transitive dependencies.

DEFINATION:

A relation is in 3NF, if it is in 2NF and all non-prime attributes of the relation are non-transitively dependent on the every key.

A relation R is in 3NF if all non prime attributes are,

1. Full functionally dependent on primary key.
2. Non-transitive dependent on every key.

A relational schema R is in 3NF, if non-trivial functional dependency X→A holds true where X is a superkey and A is a prime attribute.

CHECK WHETHER IT IS IN 3NF OR NOT

Q17. Explain transaction ,properties and states with suitable example

ANS.

Ans. : Transaction

Single SQL command is sent to database server as a query and server will reply with answer. Multiple SQL commands (DML,DRL etc.) are sent to database server which executed one after other

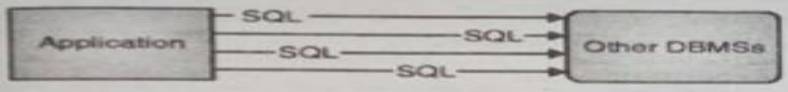


Fig. 10.1 : Executing single operation in DBMS

In place of sending one by one SQL command to server we can combine multiple operations that are logically similar and send to serve as a single logical unit called **transaction**.

Example : Transferring Rs.100 from one account to other

1. Withdraw Rs.100 from account_1
2. Deposit Rs.100 to account_2

Simple query fired on DBMS is called **SQL operation**. Collection of multiple operations that forms a single logical unit is called as **transaction**. A transaction is a sequence of one or more SQL statements that combined together to form a single logical unit of work.

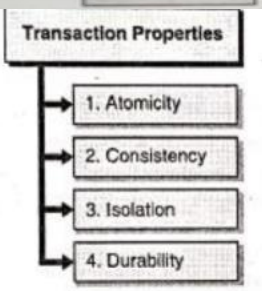


Fig. 10.2.2 : Transaction Properties

1. Atomicity

- Transaction must be treated as a single unit of operation.
- That means when a sequence of operations is performed in a single transaction, they are treated as a single large operation.

Examples :

(a) Withdrawing money from your account.

(b) Making an airline reservation.

- The term atomic means thing that cannot be divided in parts as in atomic physics.
- Execution of a transaction should be either complete or nothing should be executed at all.
- No partial transaction executions are allowed. (No half done transactions)

2. Consistency

- Consistent state is a database state in which all valid data will be written to the database.
- If a transaction violates some consistency rules, the whole transaction will be rolled back and the database will be restored to its previous consistent state with those rules.
- On the other hand, if a transaction is executed successfully then it will take the database from one consistent state to another consistent state.
- DBMS should handle an inconsistency and also ensure that the database is clean at the end of each transaction.
- Consistency means transaction will never leave your database in a half finished (inconsistent) state.
- If one part of the transaction fails, all of the pending changes made by that transaction are rolled back.

Example : Money transfer in above example

Initially in total balance in accounts A is 1000 and B is 5000 so sum of balance in both accounts is 6000 and while carrying out above transaction some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation.

3. Isolation

- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- Isolation property keeps multiple transactions separated from each other until they are completed.
- Operations occurring in a transaction (example, insert or update statements) are invisible to other transactions until the transaction commits (on transaction success) or rolls back (on transaction fails).
- For example, when a transaction changes a bank account balance other transactions cannot see the new balance until the transaction commits.
- Different isolation levels can be set to modify this default behaviour.
- Transaction isolation is generally configurable in a variety of modes. For example, in one mode, a transaction locks until the other transaction finishes.
- Even though many transactions may execute concurrently in the system. System must guarantees that, for every transactions (T_i) all other transactions has finished before transactions (T_i) started, or other transactions are started execution after transactions (T_i) finished.
- That means, each transaction is unaware of other transactions executing in the system simultaneously.

Example : Money transfer in above example.

The database is temporarily inconsistent while above transaction is executing, with the deducted total written to A and the increased total is not written to account B. If some other concurrently running transaction reads balance of account A and B at this intermediate point and computes A+B, it will observe an inconsistent value (Rs. 5900). If that other transaction wants to perform updates on accounts A and B based on the inconsistent values (Rs. 5900) that it read, the database may be left database in an inconsistent state even after both transactions have completed.

A way to avoid the problem of concurrently executing transactions is to execute one transaction at a time.

4. Durability

- The results of a transaction that has been successfully committed to the database will remain unchanged even after database fails.

- Changes made during a transaction are permanent once the transaction commits.
- Even if the database server is fails in the between transaction, it will return to a consistent state when it is restarted.
- The database handles durability by transaction log.
- Once the execution of the above transaction completes successfully, and the user will be notified that the transfer of amount has taken place, if there is no system failure in this transfer of funds.
- The durability property guarantees that, all the changes made by transaction on the database will be available permanently, although there is any type of failure after the transaction completes execution.

Q18. Explain timestamp based protocol and how timestamp-ordering protocol guarantees serializability

ANS.

(1) Introduction

- To achieve serializability order of transactions for execution can be decided in advance using its time at which transaction entered in system.
- A general method to achieve this is using time stamp ordering protocol.

(2) Concept of Timestamp

- A fixed timestamp is assigned at start of execution of the transaction. Every transaction T_j has been assigned a timestamp by database system denoted as TS (T_j).
- A transaction which has entered in system recently will have greater timestamp. If transaction T_j starts after T_i then,
- $$TS(T_i) < TS(T_j)$$
- **System clock** is used as timestamp i.e. system time when transaction T_i enters system or a **logical counter** can be used as timestamp and incremented after every assignment.
- If $TS(T_i) < TS(T_j)$, then system must ensure that serializable schedule is equivalent to a serial schedule as $< T_i, T_j >$

- Every data item X is with two timestamp values :

  (a) W-timestamp (X)

  - It denotes the largest timestamp of any transaction that executed WRITE (X) successfully on given data item X.
  - That mean it is timestamp of recent WRITE (X) operation.
  - On execution of next WRITE (X) operation (O_1), W-timestamps will be updated to new timestamp of O_1 i.e. TS (O_1).

  (b) R-timestamp (X)

  - Denotes the largest timestamp of any transaction that executed READ (X) successfully on data item (X).
  - That mean it is timestamp of recent READ (X) operation.
  - On execution of next READ (X) operation (O_1), R-timestamps will be updated to new timestamp of O_1 i.e. TS (O_1).

(3) Timestamp - ordering Protocol

- This protocol ensures any conflicting READ or WRITE is executed in order of timestamp.
- If by any reasons, if transaction is aborted then on restarting, new timestamp is assigned.

Working

(a) For transaction T_i to execute READ (X) Operation,

If TS (T_i) < W - timestamp (X)

Then T_i is trying to read value of X that is overwritten by other transaction.

W - timestamp (X) = 149 (Recent Write)			
TS	**T_i**	**T_x**	**Operations**
148	READ (X)	...	TS (T_i) < W - timestamp (X) i.e. 148 < 149 (W-Timestamp)
149	Unlock (X)	WRITE (X)	Record W - timestamp (X) = 149

So this READ is rejected (as T_x is already performing write operation on data so no other operation can be performed by any other transaction) and T_i is rolled back.

If TS (T_i) ≥ W - timestamp (X)

Then READ executed and set

R-timestamp(X) = max {TS (T_i), R-timestamp}			
TS	**T_i**	**T_x**	**Operations**
148	....	WRITE (X)	Record W - timestamp (X) = 148 (recent write)
149	READ (X)	....	TS (T_i) ≥ W - timestamp (X) i.e. 149 ≥ 148 (W-Timestamp) Record R-timestamp(X) = 149
150	....	....	....
151	READ (X)	....	TS (T_i) ≥ W - timestamp (X) i.e. 151 ≥ 148 (W-Timestamp) Record R-timestamp(X) = max{149,151} = 151

(b) If transaction T_i execute WRITE (X) Operation,

If TS (T_i) < R - timestamp (X)

Then T_i has produced value of X which is not needed now so rollback T_i.

TS	**T_i**	**T_x**	**Operations**
148	....	...	...
149	WRITE(X)	....	TS (T_i) < R - timestamp (X) i.e. 149 < 151 (R-Timestamp) Reject WRITE roll back T_i
150	....	....	....
151		READ (X)	Record R-timestamp(X) = 151 (Recent READ Operation)

If TS (T_i) < W - timestamp (X)

Then T_i is trying to write obsolete value of X, So rollback T_i

TS	T_i	T_x	Operations
148	....	...	...
149	WRITE (X)	....	TS (T_1) < W - timestamp (X) i.e. 149 < 151 (W-Timestamp) Cannot write as other transaction using data X for writing.
150	....	....	....
151		WRITE (X)	Record W-timestamp(X) = 151

(c) Otherwise, system executes WRITE (X) and sets

W-timestamp (X) = TS (T_i)

TS	T_i	T_x	Operations
150	....	....	....
151	WRITE (X)		Record W-timestamp(X) = 151

(4) Advantages

- This protocol ensures conflict serializability, as conflicting operations are processed in order of timestamp of operation.
- Ensures that it is free from deadlock.

(5) Disadvantages

- Starvation is possible for long transaction if short transaction conflicts with it causes restorative of long transaction again and again.
- It may not give recoverable schedules.