UNIT-5
DATABASE RECOVERY &TRANSACTION MANAGEMENTSYSTEM

Transaction:

Any logical work or set of works that are done on the data of a database is known as a transaction. Logical work can be inserting a new value in the current database, deleting existing values, or updating the current values in the database.

For example, adding a new member to the database of a team is a transaction.

To complete a transaction, we have to follow some steps which make a transaction successful. For example, we withdraw the cash from ATM is an example of a transaction, and it can be done in the following steps:

- Initialization if transaction
- Inserting the ATM card into the machine
- Choosing the language
- Choosing the account type
- Entering the cash amount
- Entering the pin
- Collecting the cash
- Aborting the transaction

So, in the same way, we have three steps in the DBMS for a transaction which are the following:

- Read Data
- Write Data
- Commit

PROPERTIES OF TRANSACTION(ACIDPROPERTIES):

To ensure consistency,completeness of the database in scenario of concurrent access,the following **ACID** properties can be enforced on to database.

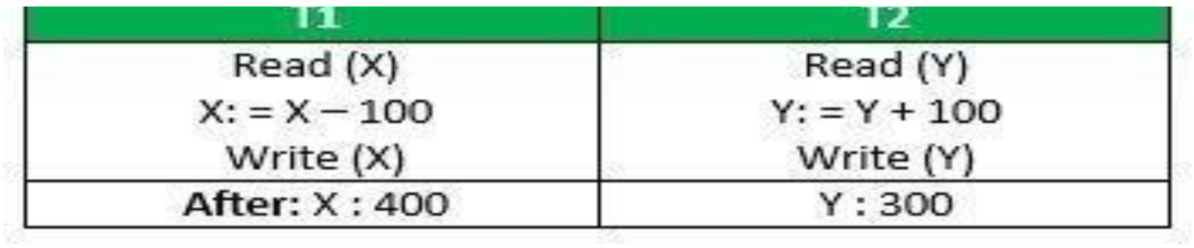
1. **Atomicity,**
2. **Consistency,**
3. **Isolation**
4. **Durability**

Atomicity:

- This property states that all of the instructions with in a transaction must be executed or none of them should be executed.
- This property states that all transactions execution must be atomic i.e. all actions should be carried out or none of the actions should be executed.

  - It involves following two operations.
    Abort:If a transaction aborts, changes made to database are not visible.
    Commit:If a transaction commits, changes made are visible. Atomicity is also known as the 'All or nothing rule'.

    Example:
  - Consider the following transaction **T** consisting of **T1** and **T2**:
  - Transfer of 100 from account **X** to account **Y**.

- 

T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

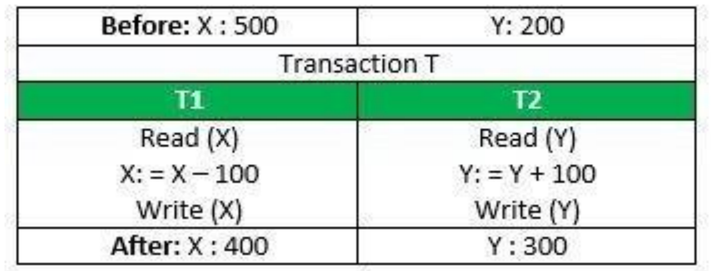
  - If the transaction fails after completion of

    T1 but beforecompletionof**T2**.(say,after

write(X) but before **write(Y)**),then amount has been deducted from **X** but not added to**Y**.Thisresults in an inconsistent database state.Therefore,the transactionmustbe executed in entirety in order to ensure correctness of database state.

Consistency:

- The database must remain inconsistence state even after performing any kind of transaction ensuring correctness of the database.
- If we execute a particular transaction in isolation(or)together with other transaction in multiprogramming environment ,the transaction should give same result in any case.
- Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency** and the DBMSassumesthatitholdsforeachtransaction.
  example:



Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

- Referring to the example above,
  The total amount before and after the transaction must be maintained. Total **before T** occurs = **500 + 200 = 700**.

  Total**afterToccurs**=**400+300=700**.
  Therefore,database is **consistent**.Inconsistency
  Occurs in case **T1** completes but **T2**
  fails.As a result T is incomplete.

Isolation:

- When executing multiple transactions concurrently & trying to access shared resources the system should create an order such that the only one transaction can access the shared resource at the sametime&release it after completion of it's execution for other transaction.
- This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state

Durability:

- This property states that once after the transaction is completed the changes that made should be permanent & should be recoverable even after system crash/power failure.
- This property ensures that once the transaction has completed execution, the updates and modificationstothedatabasearestoredinandwrittentodiskandtheypersi stevenissystem failure occurs. These updates now become permanent and are stored in a non-volatile memory.

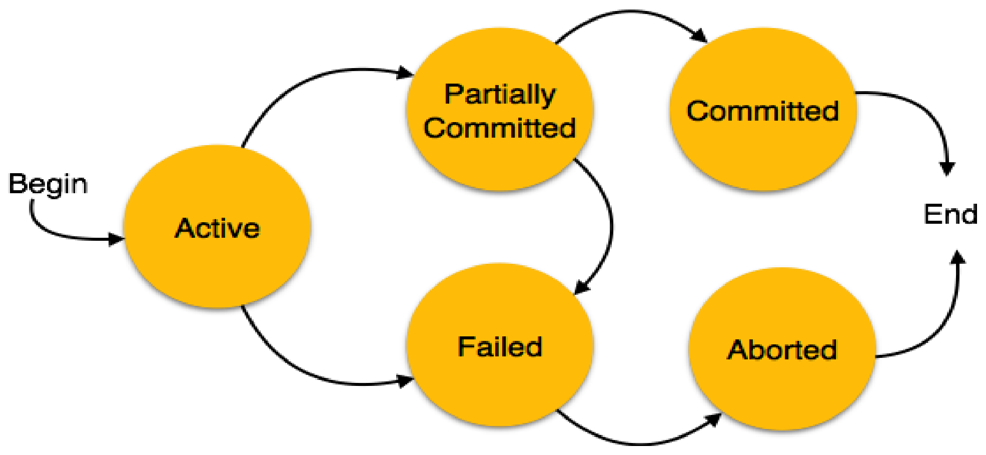
Transaction states:

Every transaction undergoes several states in its execution. A

transaction can be in any one of the following states:

1. Start/Active
2. partially committed
3. committed
4. failed
5. aborted/terminate

Transaction state diagram



- **Active -** This is the first state of transaction and here the transaction is being executed. For example, updating or inserting or deleting a record is done here. But it is still not saved to the database. When we say transaction it will have set of small steps, and those steps will be executed here.

- **Partially Committed -** This is also an execution phase where last step in the transaction is executed. But data is still not saved to the database. In example of calculating total marks, final display the total marks step is executed in this state.

- **Committed -** In this state, all the transactions are permanently saved to the database. This step is the last step of a transaction, if it executes without fail.

- **Failed -** If a transaction cannot proceed to the execution state because of the failure of the system or database, then the transaction is said to be in failed state. In the total mark calculation example, if the database is not able fire a query to fetch the marks, i.e.; very first step of transaction, then the transaction will fail to execute.

- **Aborted-**If there is any failure during the execution, then the system goes from failed to an aborted state. From an aborted state, the transaction will start its execution from a fresh start or from a newly active state.

- Implementation of Durability&Atomicity:

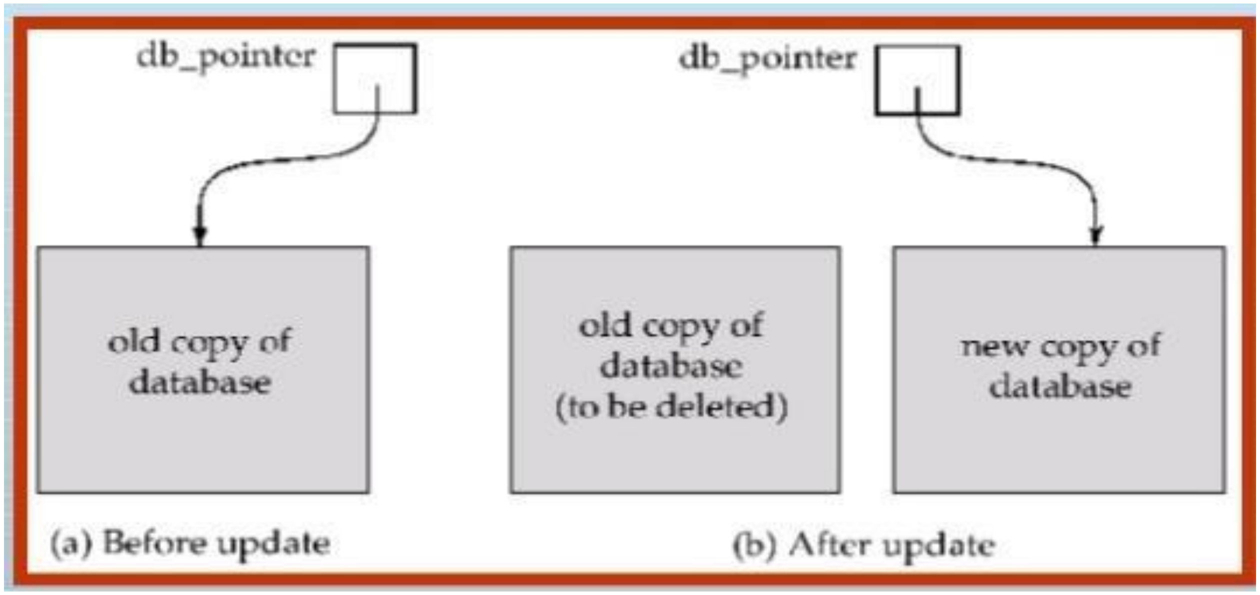
Durability and atomicity can be ensured by using Recovery manager which is available by default in every DBMS.

- We can **implement atomicity** by using

1. Shadow copying technique
2. Using recovery manager which available by default in DBMS.

1. Shadow copying technique:

1. Maintaining a shadow copy of original database & reflecting all changes to the database as a result of any transaction after committing the transaction.
2. The scheme also assumes that the database is simply a file on disk.
3. A pointer called db-pointer is maintained on disk; it points to the current copy of the database.
4. In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database.
5. All updates are done on the new database copy, leaving the original copy, the **shadow copy**, untouched.
6. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
7. If the transaction completes, it is committed as follows.
8. First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.)
9. After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.



We now consider how the technique handles transaction and system failures.

First, **consider transaction failure**. If the transaction fails at any time before db-pointer is updated, the old contents of the database are not affected. We can abort the trans- action by just deleting the new copy of the database. Once the transaction has been committed, all the updates that it performed are in the database pointed to by db- pointer. Thus, either all updates of the transaction are reflected, or none of the effects are reflected, regardless of transaction failure.

<u>WE CAN IMPLEMENT DURABILITY AMONG DATABASE USING</u>:**

1. Recoverymanager.
2. Logs

- Partial transaction should be avoided for ensuring atomicity and durability.

<u>LOGS:</u>

- Logs keep track of actions carried out by transactions which can be used for the recovery of database in case of failure.
- Logs files should be stored always on stable storage devices.
- When a transaction begins its execution it is recorded in the log as follows.

<Tn,start>

- When a transaction performs an operation it is recorded in log as follows

  <Tn,X,V1, V2>

- When a transaction finishes it's execution, it is recorded as

  <Tn , commit>

CONCURRENT EXECUTION:

Executingasetoftransactionssimultaneouslyinapreemptiveandtimesharedmethod.

InDBMSconcurrentexecutionoftransactioncanbeimplementedwithinterleaved execution.

TRANSACTION SCHEDULES:

Schedule:

- It refers to the list of actions to be executed by transaction.
- A **schedule** is a process of grouping the transactions into one and executing them in a predefined order.
- Schedule of actions can be classified into 2 types.

1. Serializable schedule/serialschedule.
2. Concurrent schedule.

1. Serial schedule:

In the serial schedule the transactions are allowed to execute one after the other ensuring correctness of data.

A schedule is called serial **schedule**,if the transactions in the schedule are defined to execute one after the other.

2. **Concurrent schedule:**

Concurrent schedule allows the transaction to be executed in inter leaved manner of execution.

Complete schedule:

It is a schedule of transactions where each transaction is committed before terminating. The example is shown below where transactions T1 and T2 terminates after committing the transactions.

Example:

T1	T2
A=1000	
Read(A)	
A=A+100	
Write(A)	Read(A)
	B=A-100
	Write(B)
	Commit
Read(B)	
Write(B)	
Commit	

SERIALIZABILITY:

A transaction is said to be **Serializable** if it is equivalent to serial schedule.

Serializability aspects are:

1. Conflict serializability.
2. View serializability.

1. Conflict serializability:

A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule.

Conflict Equivalent: Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

Conflict Serializable: A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Conflicting operations: Two operations are said to be conflicting if all below conditions are satisfied

- They belong to different transactions.
- They operation on same data item
- At Least one of them is a write operation.

It refers to two instructions of two different transactions may want to access same data to perform read/write operation.

Rules for conflict serializability:

- If two different transactions are both for read operation, then there is no conflict and can allowed to execute any order.
- If one instruction performing read operation and other instruction performing write operation there will be conflict hence instruction ordering is important.
- If both transactions performing write operation then there will be in conflict so ordering the transaction can be done.

2. View serializability:

This is another type of serializability that can be derived by creating another schedule out of an existing Schedule.

A schedule is **view serializable** if it is view equivalent to some serial schedule. Every conflict serializable schedule is view serializable, although the converse is not true.

Two schedules *S*1 and *S*2 over the same set of transactions—any transaction that appears in either *S*1 or *S*2 must also appear in the other are **view equivalent** under these conditions:

1. If *Ti* reads the initial value of object *A* in*S*1,it must also read the initial value of *A* in*S*2.
2. If *Ti* reads a value of *A* written by *Tj* in *S*1, it must also read the value of *A* written by *Tj* in *S*2.
3. For each data object *A*, the transaction (if any) that performs the final write on *A* in *S*1 must also perform the final write on *A* in *S*2.

- The above two schedules are view serializable or view equivalence, if the transactions in both schedules performs the actions in similar manner.
- The above two schedules satisfy result view equivalence if the two schedule produces the same Result after execution.

  Ex: s1:R1(A),W1(A),R2(A),W2(A),

  R1(B),W1(B),R2(B),W2(B)

Recoverability:

It refers to the process of undoing the changes made to the database in case of any transaction failure due to system crash or any other reason.

Recoverability Schedule:

Basedonwhetherrecoveryoffailuretransactionschedulesareclassifiedas

1. Irrecoverable schedules.
2. Recoverable schedules with cascade rollback.
3. Cascadeless recoverability.

1. **Irrecoverable schedules:scheduleswhichcan'tbe recovered**

- If transaction T2 read the value updated by Transaction T1 followed by write operation commit then this schedule is called Irrecoverable Schedule. If transaction1 failed before committing

Example:

T1	T1'sbufferspace	**T2**	T2'sbuffer space	database
R(A)	A=5000			A=5000
A=A-100	A=4000			A=5000
W(A)	A-4000			A=5000
		R(A)	A=4000	A=4000
		A=A+500	A=4500	A=4000
		W(A) Commit;	A=4500	A=4000
Failure point				A=4000
Commit				A=4500

Implementation of Isolation:

- When more than one instruction of several transaction are being executed concurrently by using some sharable resources , the execution of instruction of one transaction should not interrupted the execution of instruction of another transaction.
  1. **Access to sharable resources should be order by using some locking mechanism:** Where one transaction locks the sharable resource before starting it's execution& release the lock to other transaction after completion of it's execution.
  2. **Locking protocols:** Locking mechanism can be implemented by using locking protocols which defined set of standard rule based on which transaction access, sharable resources.

Transaction control commands supported with SQL:

1. Commit.
2. Save point.
3. Roll back.

Explain about usage of above 3 commands with syntaxes.

Transaction Control Language(TCL) commands are used to manage transactions in the database.

Before moving forward with TCL commands, check these topics out first:

1.COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

2.ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

3.SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

Locking system:

Advantages of Locking

- **Data Consistency:** Locking can help ensure data consistency by preventing multiple users from modifying the same data simultaneously. By controlling access to shared resources, locking can help prevent data conflicts and ensure that the database remains in a consistent state.
-
- **Isolation:** Locking can ensure that transactions are executed in isolation from other transactions, preventing interference

between transactions and reducing the risk of data inconsistencies.

-
-
- **Granularity:** Locking can be implemented at different levels of granularity, allowing for more precise control over shared resources. For example, row-level locking can be used to lock individual rows in a table, while table-level locking can be used to lock entire tables.

- **Availability:** Locking can help ensure the availability of shared resources by preventing users from monopolizing resources or causing resource starvation.

Disadvantages of Locking:

- **Overhead:** Locking requires additional overhead, such as acquiring and releasing locks on shared resources. This overhead can lead to slower performance and increased resource consumption, particularly in systems with high levels of concurrency.
-
- **Deadlocks:** Deadlocks can occur when two or more transactions are waiting for each other to release resources, causing a circular dependency that can prevent any of the transactions from completing. Deadlocks can be difficult to detect and resolve and can result in reduced throughput and increased latency.

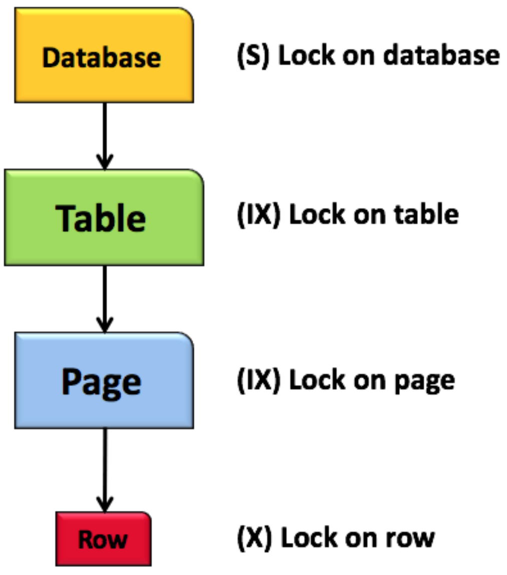
- **Reduced Concurrency:** Locking can limit the number of users or applications accessing the database simultaneously. This can lead to reduced concurrency and slower performance in systems with high levels of concurrency.

- **Complexity:** Implementing locking can be complex, particularly in distributed systems or in systems with complex transactional logic. This complexity can lead to increased development and maintenance costs.

Lock Granularity:

A database is basically represented as a collection of named data items.The size of the data item chosenas the unit of protection by a concurrency control program is called GRANULARITY. Locking can take place at the following level :

1. Database level.
2. Table level.
3. Page level.
4. Row(Tuple)level.
5. Attributes(fields)level



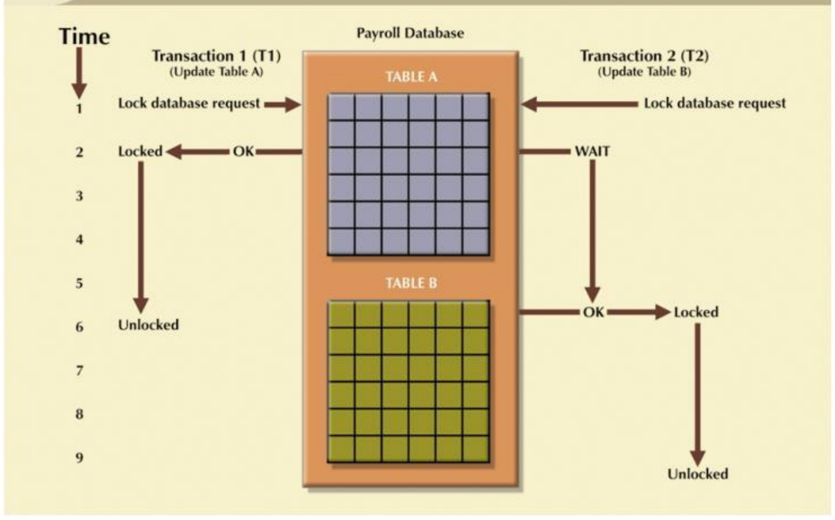
i. **Database level Locking:**

At database level locking, the entire database is locked.Thus,it prevents the use of any tables in the database by transaction T2 while transaction T1is being executed. Database level of locking is suitable for batch processes.Being very slow, it is unsuitable for on-line multi-user DBMSs.

10

Lock Granularity (continued)

FIGURE 10.3 Database-level locking sequence



Database Systems: Design, Implementation, & Management, 7th Edition, Rob & Coronel

32

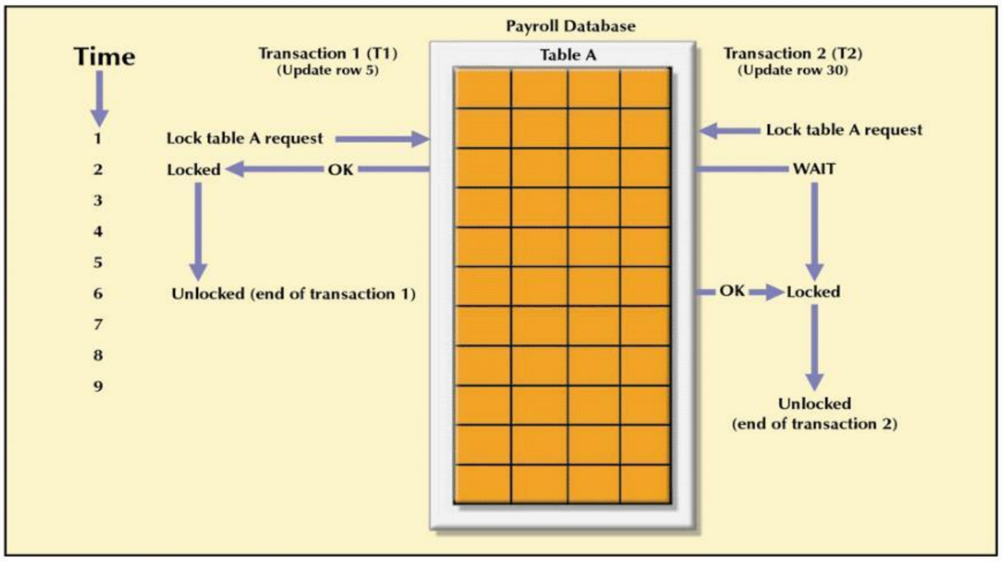
ii. **Table level Locking:**

At table level locking, the entire table is locked.Thus,it prevents the access to any row(tuple) by transaction T2 while transaction T1 is using the table. if a transaction requires access to several tables, each table may be locked.However, two transactions can access the same database as long as they access different tables. Table level locking is less restrictive than database level.Table level locks are not suitable for multi-user DBMS.

10

An Example of a Table-Level Lock

FIGURE 9.4 AN EXAMPLE OF A TABLE-LEVEL LOCK



39

iii. **Page level Locking:**

At page level locking, the entire disk-page (or disk-block) is locked. A page has a fixed size such as4K,8K,16K,32Kand so on. A table can span several pages, and a page can contain several rows (tuples) of one or more tables. Page level of locking is most suitable for multi-user DBMSs.

iv. **Row(Tuple) level Locking:**

At row level locking, particular row(or tuple)is locked. A lock exists for each row in each table of the database.The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page.The row level lock is much less restrictive than database level, table level, or page level locks. The row level locking improves the availability of data. However, the management of row level locking requires high overhead cost.

v. Attributes(fields) level Locking:

At attribute level locking, particular attribute(orfield) is locked.Attribute level locking allows concurrent transactions to access the same row, as long as they require the use of different attributes within the row. The attribute level lock yields the most flexible multi-user data access. It requires a high level of computer overhead.

Locking protocols:

1.Simple lock based protocol

2.Conservative(or)pre-claimlocking protocol.

3.2-phase locking protocol
  a).Strict 2 phase locking protocol

  b).Rigorous 2phase locking protocol

1.Simple lock based protocol:

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation isperformed.Transactions may unlock the data item after completing the‘write’ operation.

Problems with simple locking are:

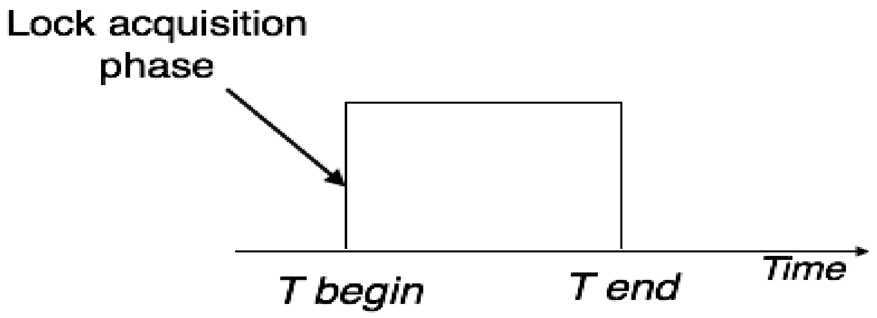
1. deadlocks

2. starvation

2.Conservative(or)pre-claimlockingprotocol:

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand.

If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



3.2-phaselockingprotocol(2PL):

This locking protocol divides the execution phase of a transaction into three parts.

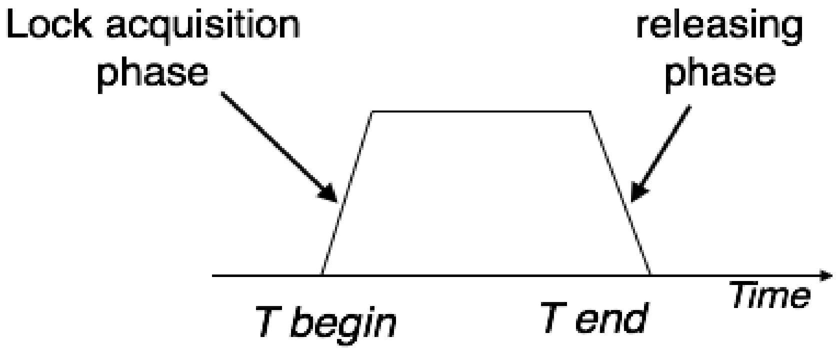
- In the first part,when the transaction starts executing,it seeks permission for the locks it requires.
- The second part is where the transaction acquires all the locks.As soon as the transaction releases its first lock, the third phase starts.

- In third phase, the transaction cannot demand any newlocks;it only releases the acquired locks.

This protocol can be divided into two phases,

1. **In Growing Phase,** a transaction obtains locks, but may not release any lock.

2. **In Shrinking Phase,** a transaction may release locks, but may not obtain any lock.



Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive(write) lock, a transaction must first acquire a shared(read)lock and then upgrade it to an exclusive lock.

Types of Two–Phase Locking Protocol

Following are the types of two–phase locking protocol:

1. Strict Two– Phase Locking Protocol
2. Rigorous Two–Phase Locking Protocol
3. Conservative Two–Phase Locking Protocol

StrictTwo-PhaseLocking:

1. If a transaction want to read any value it can refers to a shared lock.
2. If a transaction to write any particular value it can refers to an exclusive locks.

3. A shared lock acquire by multiple transaction at sametime.
4. An exclusive lock can be requested by only one transaction at a time on any data item.
5. StrictTwo-Phase Locking Protocol avoids cascaded rollbacks.
6. It ensures that if data is being modified by one transaction,then other transaction cannot read it until first transaction commits.

Phases in strict 2phase locking:

phase1: The first phase of Strict-2PL is same as 2PL i.e.when the transaction starts executing,it seeks permission for the locks it requires.

phase2: After acquiring all the locks in the first phase,the transaction continues to execute normally.

phase3: But in contrast to 2PL,Strict-2PL does not release a lock after using it.Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

Note: It releases only all exclusive locks but not shared locks after a transaction is committed. This protocol is not free from deadlocks

RigorousTwo-PhaseLocking

- Rigorous Two–Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

- It releases all the locks including shared and exclusive locks after committingthe transactions.

- It considers the order of commit among transaction executions.

ConservativeTwo-PhaseLockingProtocol

- ConservativeTwo–PhaseLockingProtocolisalsocalledasStaticTwo–PhaseLocking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions.

The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order. The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Example:

Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.
Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:

Starvation is possible if the same transaction is restarted and continually aborted

Validation Based Protocol

Validation based Protocol in DBMS also known as Optimistic Concurrency Control Technique is a method to avoid concurrency in transactions. In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
The Validation based Protocol is performed in the following three phases:

1. Read Phase
2. Validation Phase

3. Write Phase

Read Phase

In the Read Phase, the data values from the database can be read by a transaction but the write operation or updates are only applied to the local data copies, not the actual database.

Validation Phase

In Validation Phase, the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.

Write Phase

In the Write Phase, the updates are applied to the database if the validation is successful, else; the updates are not applied, and the transaction is rolled back.

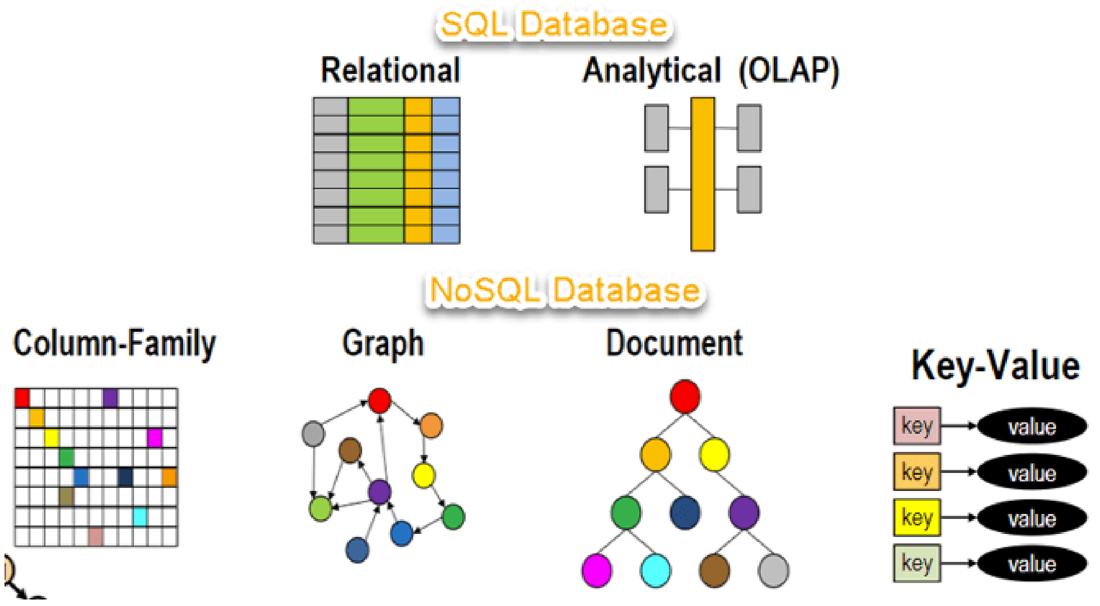
What is NoSQL?

NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale.

The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

NoSQL database stands for “Not Only SQL” or “Not SQL.”. Carl Strozz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide rangedatabase technologies that can store structured, semi-structured, unstructured and polymorphic data. Let’s understand about NoSQL with a diagram in this NoSQL database tutorial:

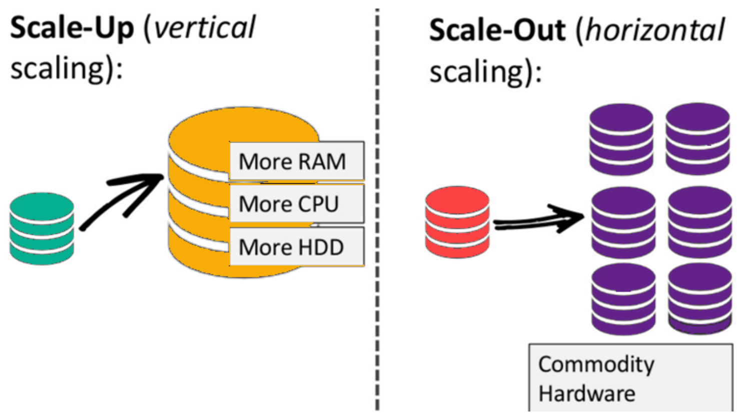


Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could “scale up” our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as “scaling out.”


Scale-Up (*vertical* scaling):
More RAM
More CPU
More HDD
Scale-Out (*horizontal* scaling):
Commodity Hardware

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Brief History of NoSQL Databases

- 1998- Carlo Strozz use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

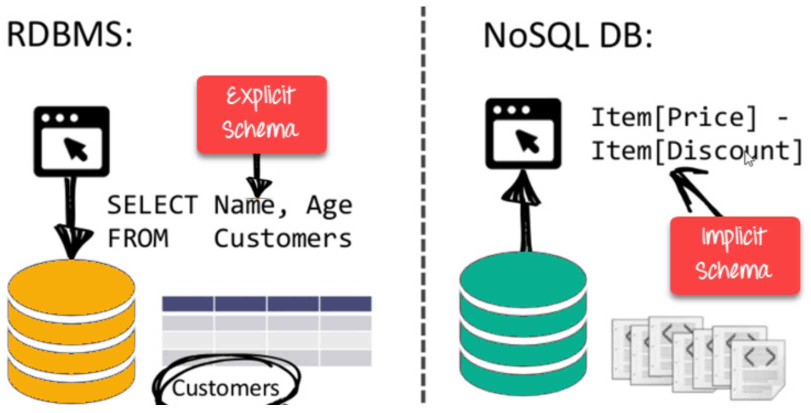
Features of NoSQL

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn’t require object-relational mapping and data normalization
- No complex features like query languages, query planners,referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



NoSQL is Schema-Free

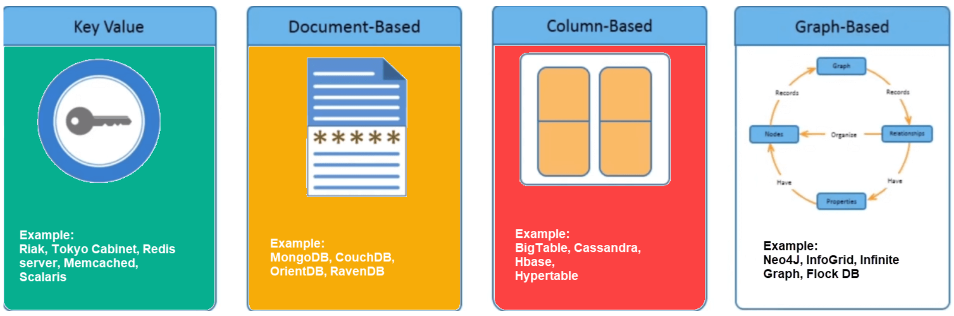
Types of NoSQL Databases

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented.

Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented



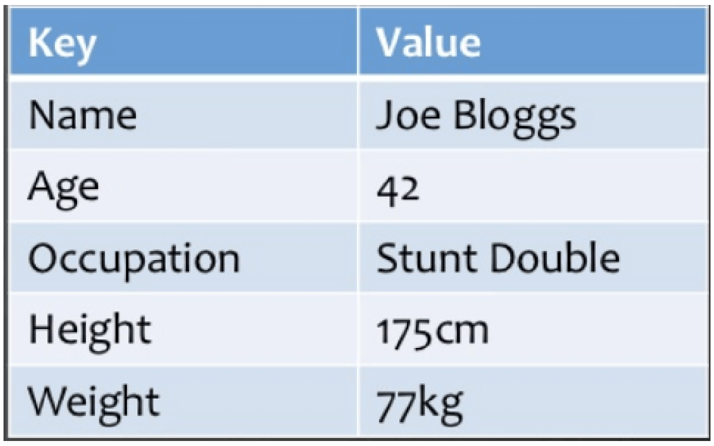
Key Value
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris
Document-Based
Example:
MongoDB, CouchDB, OrientDB, RavenDB
Column-Based
Example:
BigTable, Cassandra, Hbase, Hypertable
Graph-Based
Graph
Records
Records
Nodes
Organize
Relationships
Have
Have
Properties
Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

1.*<u>Key Value Pair Based</u>*

Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.

Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON(JAVASCRIPT OBJECT NOTATION) which is used instead of XML, BLOB(Binary Large Objects), string, etc.

For example, a key-value pair may contain a key like “Website” associated with a value like “AMAZON”.



Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

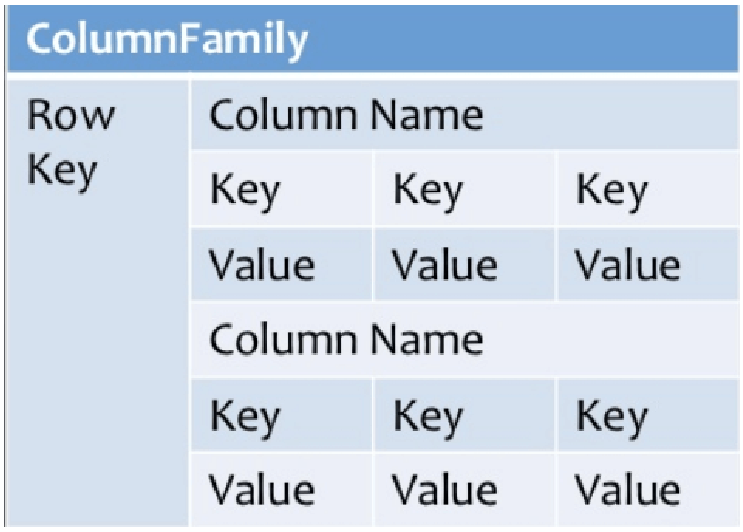
It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon’s Dynamo paper.

Companies using this s/w->twitter

2.Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.



ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column based NoSQL database

They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

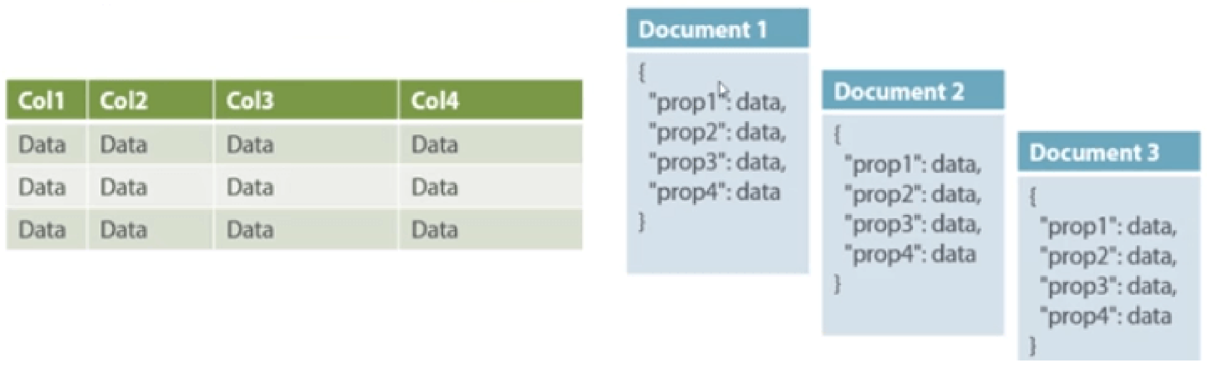
Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

Ex->companies->netflix.

3.Document-Oriented

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Relational Vs. Document

In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

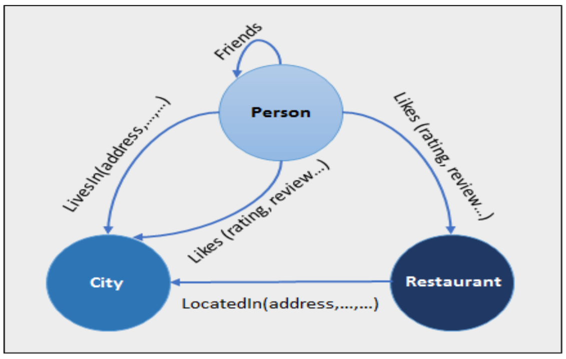
The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon Simple DB, Couch DB, Mongo DB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

Ex->uber

4.Graph-Based

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.


Friends
Person
LivesIn(address,......)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)

Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for social networks, logistics, spatial data.

Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

Ex->walmart

DATABASE RECOVERY IN DBMS AND ITS TECHNIQUES

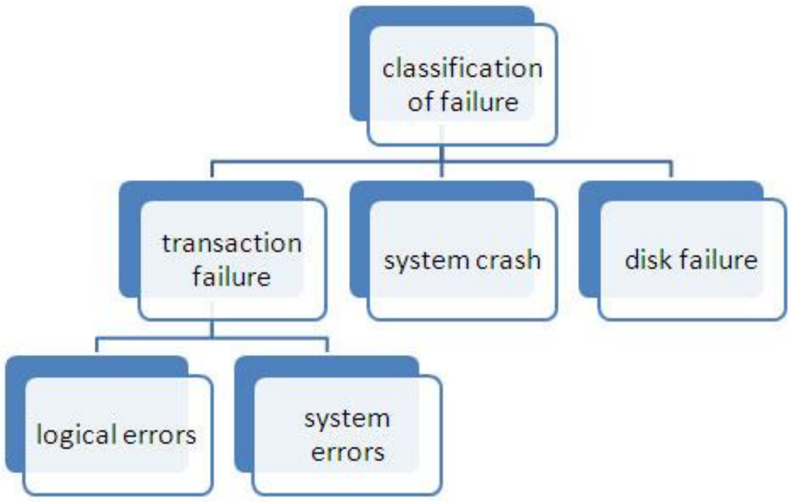
DATABASE RECOVERY IN DBMS AND ITS TECHNIQUES: There can be any case in database system like any computer system when database failure happens. So data stored in database should be available all the time whenever it is needed.

So Database recovery means recovering the data when it get deleted, hacked or damaged accidentally. So database recovery and database recovery techniques are must in DBMS. So database recovery techniques in DBMS are given below.

Classification of failure:

To see wherever the matter has occurred, we tend to generalize a failure into numerous classes, as follows:

- Transaction failure
- System crash
- Disk failure



Types of FailureS

1. **Transaction failure:** A transaction needs to abort once it fails to execute or once it reaches to any further extent from wherever it can't go to any extent further. This is often known as transaction failure wherever solely many transactions or processes are hurt. The reasons for transaction failure are:

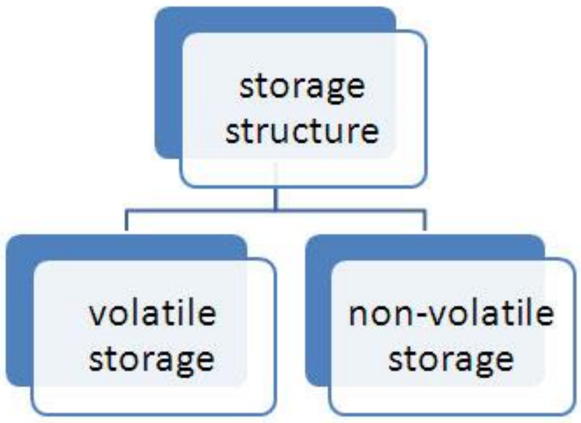
- Logical errors
- System errors

1. **Logical errors:** Where a transaction cannot complete as a result of its code error or an internal error condition.
2. **System errors:** Wherever the information system itself terminates an energetic transaction as a result of the DBMS isn't able to execute it, or it's to prevent due to some system condition. to Illustrate, just in case of situation or resource inconvenience, the system aborts an active transaction.

3. **System crash:** There are issues – external to the system – that will cause the system to prevent abruptly and cause the system to crash. For instance, interruptions in power supply might cause the failure of underlying hardware or software package failure. Examples might include OS errors.
4. **Disk failure:** In early days of technology evolution, it had been a typical drawback wherever hard-disk drives or storage drives accustomed to failing oftentimes. Disk failures include the formation of dangerous sectors, un reachability to the disk, disk crash or the other failure, that destroys all or a section of disk storage.

Storage structure:

Classification of storage structure is as explained below:



Classification Of Storage

1. **Volatile storage:** As the name suggests, a memory board (volatile storage) cannot survive system crashes. Volatile storage devices are placed terribly near to the CPU; usually, they're embedded on the chipset itself. For instance, main memory and cache memory are samples of the memory board. They're quick however will store a solely little quantity of knowledge.

2. **Non-volatile storage:** These recollections are created to survive system crashes. they're immense in information storage capability, however slower in the accessibility. Examples could include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity:

When a system crashes, it should have many transactions being executed and numerous files opened for them to switch the information items.

Transactions are a product of numerous operations that are atomic in nature. However consistent with ACID properties of a database, atomicity of transactions as an entire should be maintained, that is, either all the operations are executed or none.

When a database management system recovers from a crash, it ought to maintain the subsequent:

- It ought to check the states of all the transactions that were being executed.
- A transaction could also be within the middle of some operation; the database management system should make sure the atomicity of the transaction during this case.
- It ought to check whether or not the transaction is completed currently or it must be rolled back.
- No transactions would be allowed to go away from the database management system in an inconsistent state.

There are 2 forms of techniques, which may facilitate a database management system in recovering as well as maintaining the atomicity of a transaction:

- Maintaining the logs of every transaction, and writing them onto some stable storage before truly modifying the info.
- Maintaining shadow paging, wherever the changes are done on a volatile memory, and later, and the particular info is updated.

Log-based recovery Or Manual Recovery):

Log could be a sequence of records, which maintains the records of actions performed by dealing. It’s necessary that the logs area unit written before the particular modification and hold on a stable storage media, that is fail 0safe. Log-based recovery works as follows:

- The log file is unbroken on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log regarding it.

Recovery with concurrent transactions (Automated Recovery):

When over one transaction is being executed in parallel, the logs are interleaved. At the time of recovery, it’d become exhausting for the recovery system to go back all logs, and so begin recovering. To ease this example, the latest package uses the idea of ‘checkpoints’. Automated Recovery is of three types

Deferred Update Recovery

Immediate Update Recovery

Shadow Paging

Checkpoint:

- The checkpoint is a type of mechanism where all the previous logs are removed from the system 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 the transaction, the log files will be created.

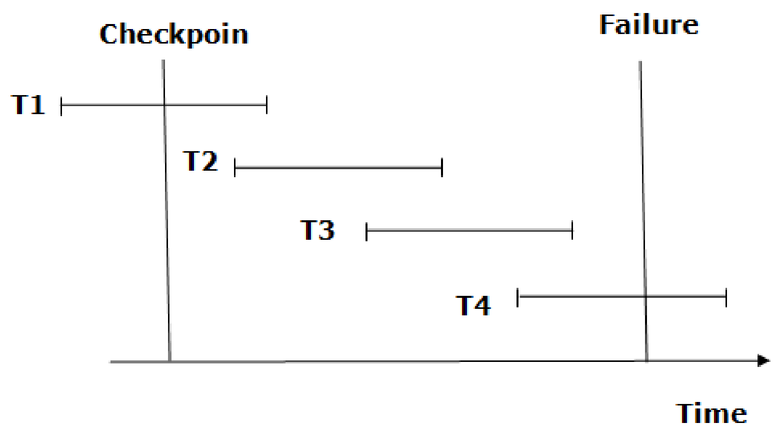
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file.

- 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 consistent state, 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 the transactions 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 the transaction 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.

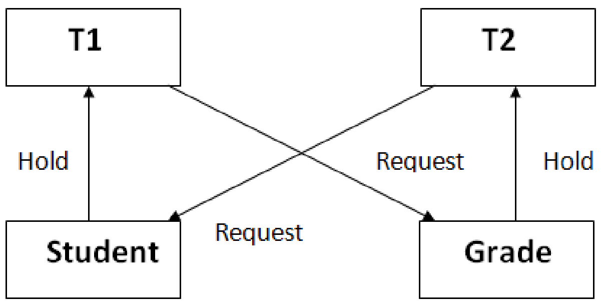


Figure: Deadlock in DBMS

1.Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the database rather 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 method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

2.Deadlock Detection

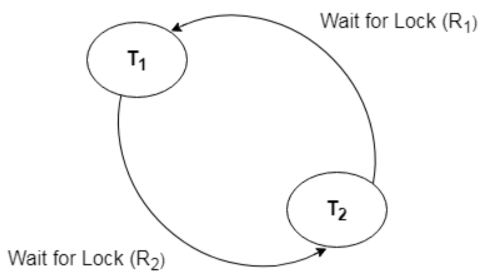
In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

ADVERTISEMENT

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 waiting for 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:



3.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 that transaction 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. It allows 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 T2 then 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 the older 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 is waiting 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 but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

Cloud database:

A cloud database is a database that is hosted and managed in the cloud. This means that the data is stored and accessed over the internet, rather than on a physical, on-premises server. Cloud databases have a few trademark features: First and foremost, they can be accessed from any location as long as you have an internet connection—making them ideal for remote teams and businesses that require on-the-go data access.

Types of cloud databases

Not all cloud databases are created equally. Cloud databases can be divided into four main categories: relational cloud databases, No SQL cloud databases, cloud data warehouses, and HTAP systems.

1. *Relational cloud database*
2. *NoSQL cloud database*
3. *Cloud data warehouse*
4. *HTAP*

1.*Relational cloud database*

Relational cloud databases, also known as SQL databases, are built on the traditional relational database model. These databases use a structured query language (SQL) to manage and manipulate data. Relational cloud databases are ideal for structured data, such as retail analytics data related to transactions, inventory, or customer information. Examples of relational cloud databases include Amazon Relational Database Service (RDS), Microsoft Azure SQL Database, and Google Cloud SQL.

2.NoSQL cloud database

NoSQL cloud databases, on the other hand, are non-relational databases. These databases are designed to handle unstructured data, such as social media posts, log files, and user-generated content. NoSQL cloud databases use different data models, such as key-value, document, graph, or column-family stores. Examples of NoSQL cloud databases include Amazon DynamoDB, MongoDB Atlas, and Google Cloud Datastore.

3.Cloud data warehouse

A cloud data warehouse is designed for big data processing and analytics. It can store vast amounts of structured and unstructured data from multiple sources, such as social media, sensors, and IoT devices. Examples of cloud data warehouses include Amazon Redshift, Google BigQuery, Microsoft Azure Synapse Analytics, and Snowflake.

4.HTAP(Hybrid Transactional/Analytical Processing)

HTAP (Hybrid Transactional/Analytical Processing) is a database architecture that combines both transactional and analytical processing capabilities in a single system. This approach allows organizations to perform real-time analytics on transactional data without the need for a separate data warehouse or data mart.

In a traditional OLTP (Online Transaction Processing) system, the focus is on processing transactions and ensuring data consistency and integrity. However, this type of system is not optimized for analytical queries that require complex aggregations or data manipulations. As a result, organizations often need a separate OLAP (Online Analytical Processing) system to perform these types of queries.

In contrast, an HTAP system combines both OLTP and OLAP capabilities in a single system. This allows organizations to perform both transactional processing and real-time analytics on the same data, without the need for data movement or synchronization.

HTAP systems typically use in-memory processing and columnar storage to achieve high performance for both transactional and analytical workloads. They also often use techniques such as indexing, query optimization, and data compression to further improve performance.