Data & Information
Data becomes **information** when it is processed, turning it into something meaningful.

Database
A Database is a collection of related data organised in a way that data can be easily accessed, managed and updated.

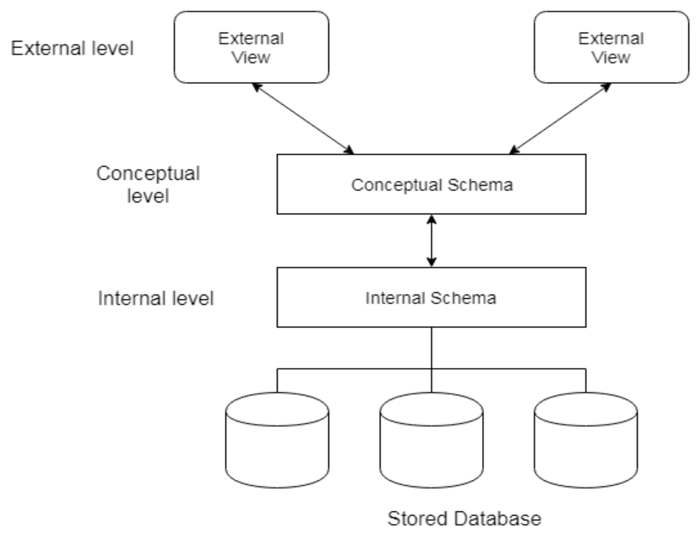
DBMS
A DBMS is a system that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily.
DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.

Database system vs files system
There are following differences between DBMS and File system:

DBMS	File System
DBMS is a collection of data. In DBMS, user is not required to write the procedures.	File system is a collection of data. In this system, user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides crash recovery mechanism i.e. DBMS protects user from the system failure.	File system doesn't have crash mechanism i.e. if the system crashes while entering some data then the content of the file is lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under file system.
DBMS contains wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like: redirecting the file while other deleting some information or updating some information.

Characteristics of DBMS
1. data store in table
2. reduced redundancy
3. data consistency
4. support multiple user and concurrent access
5. query language
6. security

Three schema Architecture
The three schema architecture is used to separate the user applications and physical database. The three schema architecture contains three-levels. It breaks the database down in three different categories.



1. **Internal Level**
   - Internal level has an internal schema which describes the physical storage structure of the database.
   - Internal schema is also known as physical schema.
   - It uses the physical data model. It is used to define that how the data will be stored in a block.
   - The physical level is used to describe complex low level data structures in detail.
2. **Conceptual Level**
   - Conceptual schema describes the design of database at the conceptual level. Conceptual level is also known as logical level.
   - The conceptual schema describes the structure of whole database.
   - The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
   - In the conceptual level, internal details such as implementation of data structure is hidden.
   - Programmers and database administrators work at this level.
3. **External Level**
   - At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe different view of the database.
   - External schema is also known as view schema.
   - Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
   - The view schema describes the end user interaction with database systems

Advantages of DBMS

- Segregation of applicaion program.
- Minimal data duplicacy or data redundancy.
- Easy retrieval of data using the Query Language.
- Reduced development time and maintainance need.

Components of DBMS

The database management system can be divided into five major components, they are:

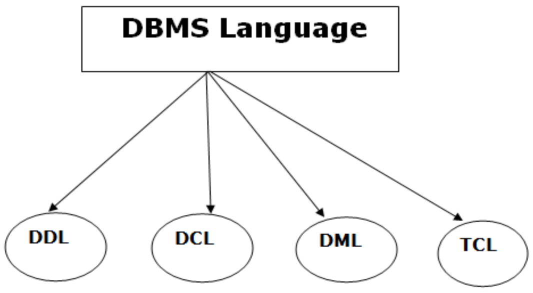
1. Hardware
2. Software
3. Data
4. Procedures
5. Database Access Language

DBMS Database Models

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. there are other models too:

- ***Hierarchical Model***
- ***Network Model***
- ***Entity-relationship Model*** (E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand. This model is good to design a database, which can then be turned into tables in relational model)
- ***Relational Model*** (In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field).

Types of Database Language



1. Data Definition Language

- DDL stands for Data Definition Language. It is used to define database structure or pattern.
- Some task come under the DDL :- **Create,alter,drop,truncate,rename,comment**

2. Data Manipulation Language

- DML stands for Data Manipulation Language. It is used for accessing and manipulating data in a database. It handles user requests.
- Some task come under the DDL :- **select ,inset,update,delete,merge,call,explain plan,lock table.**

3. Data Control Language

- DCL stands for Data Control Language. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has roll back parameters.
- Some task come under the DDL :-
- **Grant**: It is used to give user access privileges to a database.
- **Revoke**: It is used to take back permissions from the user.

There are the following operations which has authorization of Revoke:
CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped together into logical transaction.
Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit

ER MODEL

Concepts of ER Model in DBMS

The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.

Entity and Entity Set

An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.
If a Student is an Entity, then the complete dataset of all the students will be the Entity Set.

ER MODEL Keys

Following are the types of Keys:

1. Super Key
2. Candidate Key
3. Primary Key

Attribute

1. composite & simple attribute
2. single value & multivalue attribute
3. stored & derived attribute Ex- (age,date)
4. complex attribute

Relationship

- degree (how many attribute participate in relation)
- cordinality (a entiry of attribute participate in how many relation)
- participation (all the entiry participating or not)

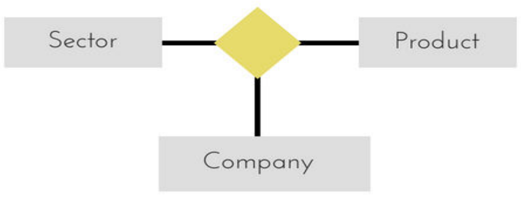
Type of relationship

a. Binary Relationship

- one to one
- one to many or many to one
- many to many

b. recursive relationship(employee (superviser) employee)

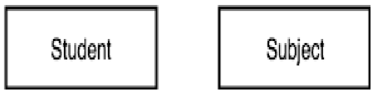
c. Ternary Relationship



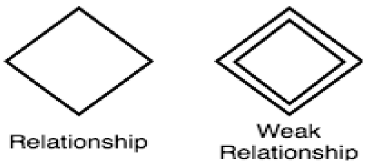
- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

ENTITY

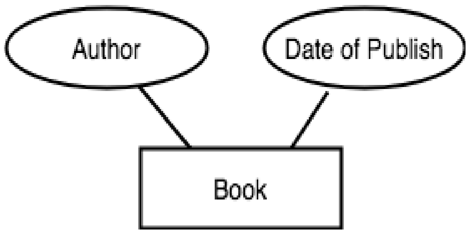
Kundan Thakur



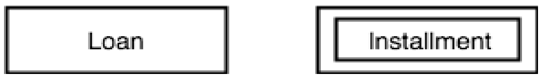
Relationships between Entities - Weak and Strong



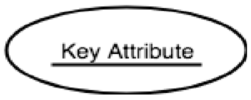
Attributes for any Entity



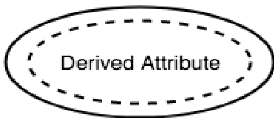
Weak Entity
Weak entity is an entity that depends on another entity. Weak entity doesn't have any key attribute of its own. Double rectangle is used to represent a weak entity



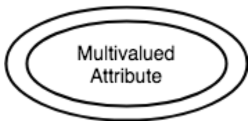
Key Attribute for any Entity



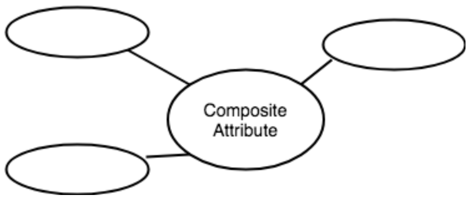
Derived Attribute for any Entity
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth



Multivalued Attribute for any Entity



Composite Attribute for any Entity



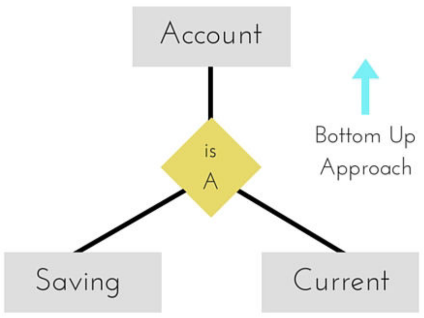
Reduction of ER diagram into Table

https://www.javatpoint.com/dbms-reduction-of-er-diagram-into-table

Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

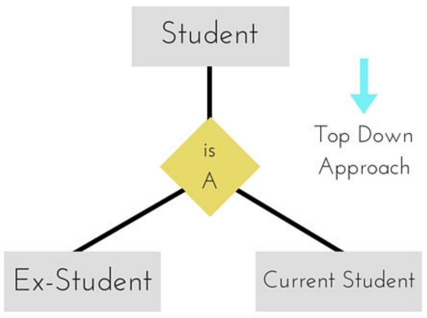
It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a super-class.



For example, **Saving** and **Current** account types entities can be generalised and an entity with name **Account** can be created, which covers both.

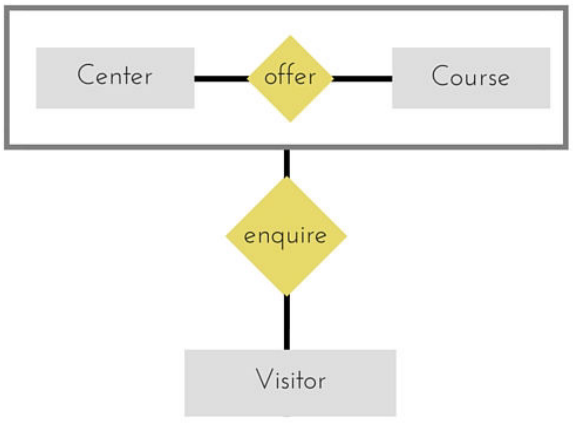
Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.



Aggregration

Aggregration is a process when relation between two entities is treated as a **single entity**.



In the diagram above, the relationship between **Center** and **Course** together, is acting as an Entity, which is in relationship with another entity **Visitor**. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

Relation Schema

A relation schema describes the structure of the relation, with the name of the relation(name of table), its attributes and their names and type.

Introduction to Database Keys

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.
A Key can be a single attribute or a group of attributes, where the combination may act as a key.

1. Super Key

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

2. Candidate Key

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table

3. Primary Key

Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

4. Foreign key

Foreign keys are the column of table which is used to point to the primary key of another table.
In a company, every employee works in a specific department and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
We add primary key of DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table. Now in the EMPLOYEE table, Department_Id is the foreign key and both the tables are related.

5. Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite**

key. But the attributes which together form the **Composite key** are not a key independentely or individually.

6. Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

7. Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table

8. Non-prime Attributes

Non-prime Attributes are attributes other than **Primary Key attribute(s).**

Join Operations: *https://www.javatpoint.com/dbms-join-operation*

Integrity Constraints

- **Integrity constraints are rules that are to be applied on database columns to ensure the validity of data.**
- Integrity constraints ensure that the data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.
- Every relation in a relational database model should abide by or follow a few constraints to be a valid relation, these constraints are called as **Relational Integrity Constraints.**

Types of Integrity Constraint

- **Domain Integrity constraint**
- **Entity integrity constraint**
- **Referential Integrity constraint**

1. **Domain constraints**

- The domain integrity constraints are used to impose restrictions on some particular column. Thus they affect the domain value. These can be enforced in the form of the following-

  Check value
  Default value

  Check is used to impose certain checks like checking if a value is greater than or lesser than a particular value etc. thus the upper and the lower limit can be set.

  Default value is the value to be provided in case no value is provided by the user. We can set a default value for any column depending on its data-type

2. **Entity integrity constraints**

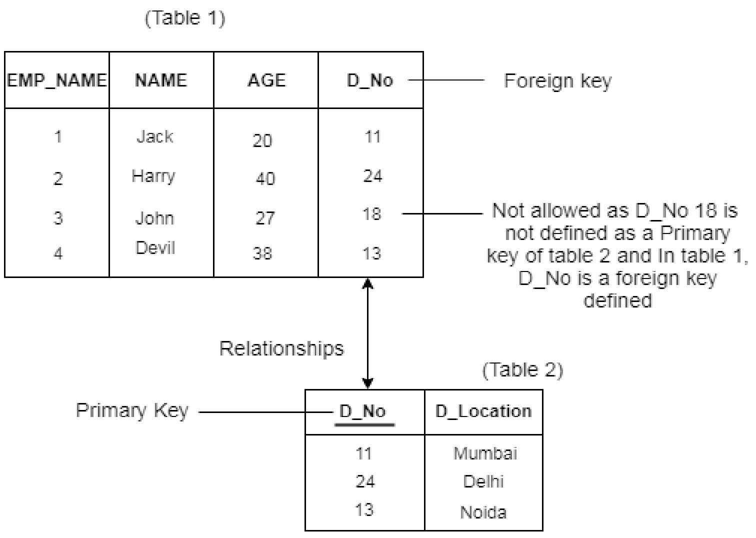
- The entity integrity constraint states that primary key value can't be null.
- The constraint here is to have a unique value for each row in the column or a group of columns it is applied to. This attribute is essential when a particular record or row of data is to be accessed. It can be accessed using the entity integrity constraint by supplying the unique value and accessing the entire

record.

There are two such constraints which ensure uniqueness of data. They are-
Primary key
Unique keyword

- This is because the primary key value is used to identify individual rows in a relation and if the primary key has null value then we can't identify those rows.
- A table can contain null value other than primary key field.

3. **Referential Integrity Constraints**
- Referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2 then every value of the Foreign Key in Table 1 must be null or be available in Table 2.


(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

functional dependency
A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B.

Attribute Closure
Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

Normalization

Normalization of Database
Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anamolies

Problems Without Normalization
If a table is not properly normalized and have data redundancy then it will take extra memory space. Insertion, Updation and Deletion Anamolies are very frequent if database is not normalized.

understand these anomalies let us take an example of a Student table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.
Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
These scenarios are nothing but **Insertion anomalies.**

Updation Anomaly

What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly**.**

Deletion Anomaly

In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

First Normal Form (1NF)

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Third Normal Form (3NF)

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

Boyce and Codd Normal Form (BCNF)

1. R must be in 3rd Normal Form
2. for each functional dependency (X → Y), X should be a super Key.

Denormalization in Databases

Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization.

In a traditional normalized database, we store data in separate logical tables ad attempt to minimize redundant data. We may strive to have only one copy of each piece of data in database.
For example, in a normalized database, we might have a Courses table and a Teachers table.Each entry in Courses would store the teacherID for a Course but not the teacherName. When we need to retrieve a list of all Courses with the Teacher name, we would do a join between these two tables.

In some ways, this is great; if a teacher changes is or her name, we only have to update the name in one place.

The drawback is that if tables are large, we may spend an unnecessarily long time doing joins on tables.

Denormalization, then, strikes a different compromise. Under denormalization, we decide that we're okay with some redundancy and some extra effort to update the database in order to get the efficiency advantages of fewer joins.

Pros of Denormalization:-

1. Retrieving data is faster since we do fewer joins
2. Queries to retrieve can be simpler(and therefore less likely to have bugs), since we need to look at fewer tables.

Cons of Denormalization:-

1. Updates and inserts are more expensive.
2. Denormalization can make update and insert code harder to write.
3. Data may be inconsistent . Which is the "correct" value for a piece of data?
4. Data redundancy necessities more storage

http://placement.freshersworld.com/technical-interview-questions-questions-and-answers/dbms/33111829

TRANSACTION

Transaction

- 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 perform operations for accessing the contents of the database.

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 Write(X) is used to write the value back to the database from 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 transaction is 4000.
- First operation reads X's value from database and stores it in a buffer.
- Second operation will decrease the value of X by 500. So buffer will contain 3500.
- Third operation will write the buffer's value to database. So X's final value will be 3500.

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.

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(ACID)

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

Property of Transaction

Atomicity

- It states that all operations of a transaction be completed, if not, the transaction is aborted.
- There is no midway i.e. transaction cannot occur partially. Each transaction is treated as one unit and either runs to completion or is not executed at all.

Atomicity involves 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.

Consistency

- The integrity constraints are maintained so that database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- Transaction is used to transform the database from one consistent state to another consistent 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, database is consistent. In case when T1 is completed but T2 fails then inconsistency will occur.

Isolation

- It is used to show that the data used at the time of execution of transaction cannot be used by second

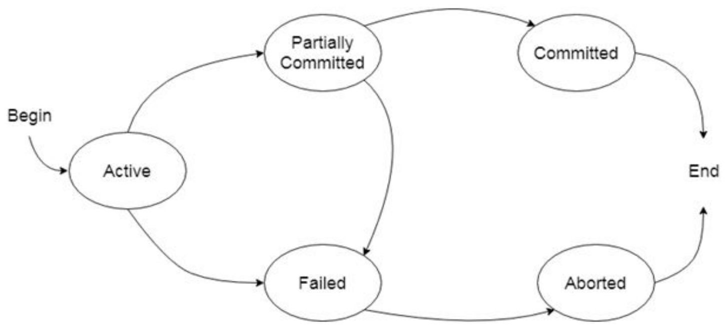
transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and is using the data item X then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. 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 consistent state. That consistent state cannot be lost, even in the event of 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

- Active state is the first state of the every transaction. In this state, transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation but the data is still not saved to the database.
- In the total mark calculation example, final display of the total marks step is executed in this 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 said to be in failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks then the transaction will fail to execute.

Aborted

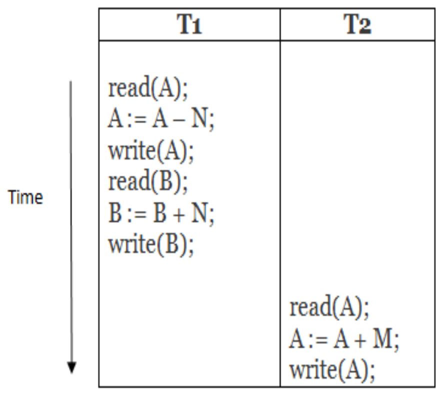
- If any of the checks fails and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into consistent state.

- 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 two operations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
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.

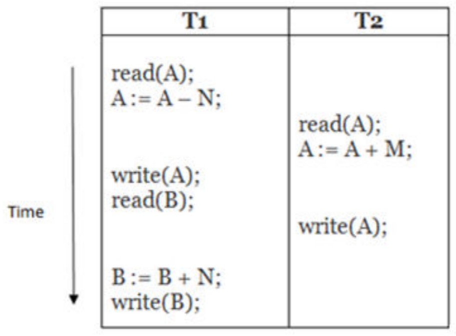
(a)



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

(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

1. Serial Schedule

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.

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

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 operations of the transactions.
- In the given figure (c) , Schedule C are the non-serial schedule. It has interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow transaction to execute concurrently without interfering one another.
- It identifies which schedules are correct when executions of transaction have interleaving of their operations.

- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

Testing of Serializability

Serialization Graph is used to test 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 set of vertices and E consists 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)

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

The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.
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, it can transform into a serial schedule.
- 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

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 contains the same set of transaction.
2. If each pair of conflicting operations is ordered in the same way.

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 .

View Serializability

- A schedule is 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 is 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

Initial read of both schedules must be same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 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

Final write must be same between both the schedules. In schedule S1, If a transaction T1 updates A at last then in S2, final writes operations should also 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 schedulesis view equal because Final write operation in S1 is done by T3 and in S2, 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, 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:

Schedule S1
Step 1: final updation on data items
In both schedules S and S1, there is no read except the initialread that's why we don't need to check that condition.
Step 2: Initial Read
Initial read operation in S is done by T1 and in S1, it is also done by T1.
Step 3: Final Write
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 the all three conditions so, we don?t need to check other schedule.
Hence, view equivalent serial schedule is:
T1→T2→T3.

Recoverability of Schedule

Sometimes a transaction may not execute completely due to 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 failed transaction. So we also have to rollback those transactions.

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

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

Concurrency Control

- In the concurrency control, the multiple transactions can be executed simultaneously.
- It may affect the transaction result. It is highly important to maintain the order of execution of those transactions.

Problems of concurrency control

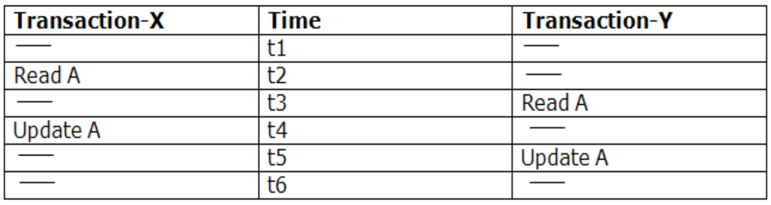
Several problems can occur when concurrent transactions are executed in an uncontrolled manner. Following are the three problems in concurrency control.

1. Lost updates

2. Dirty read
3. Unrepeatable read

1. Lost update problem

- When two transactions that access the same database items contain their operations in a way that makes the value of some database item incorrect then the lost update problem occurs.
- If two transactions T1 and T2 read a record and then update it, the effect of updation of first record will be overwritten by the second update.



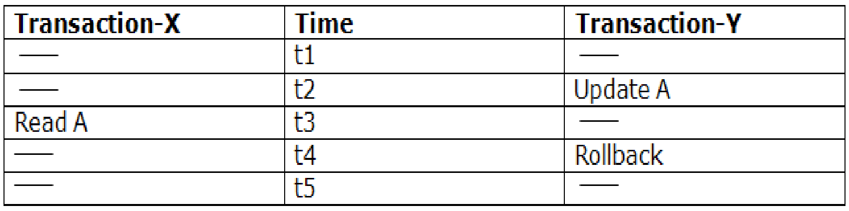
Transaction-X	Time	Transaction-Y
—	t1	—
Read A	t2	—
—	t3	Read A
Update A	t4	—
—	t5	Update A
—	t6	—

At time t2, transaction-X reads A's value.

- At time t3, Transaction-Y reads A's value.
- At time t4, Transactions-X writes A's value on the basis of the value seen at time t2.
- At time t5, Transactions-Y writes A's value on the basis of the value seen at time t3.
- So at time T5, the update of Transaction-X is lost because Transaction'y overwrites it without looking at its current value.
- Such type of problem is known as Lost Update Problem as update made by one transaction is lost here.

2. Dirty Read

- The dirty read occurs in the case when one transaction updates an item of the database and then the transaction fails for some reason. The updated database item is accessed by another transaction before it is changed back to the original value.
- A transaction T1 updates a record which is read by T2. If T1 aborts then T2 now has values which have never formed part of the stable database.



Transaction-X	Time	Transaction-Y
—	t1	—
—	t2	Update A
Read A	t3	—
—	t4	Rollback
—	t5	—

At time t2, transaction-Y writes A's value.

- At time t3, Transaction-X reads A's value.
- At time t4, Transactions-Y rollbacks. So, it changes A's value back to that of prior to t1.
- So, Transaction-X now contains a value which has never become part of the stable database.
- Such type of problem is known as Dirty Read Problem, as one transaction reads a dirty value which has not been committed.

3. Inconsistent Retrievals Problem

- Inconsistent Retrievals Problem is also known as **unrepeatable read**. When a transaction calculates some summary function over a set of data while other transaction sare updating the data then Inconsistent Retrievals Problem occurs.
- The transaction can read some data after they are changed and other data before they are changed due to this the inconsistent result will create.
- A transaction T1 reads a record and then does some other processing during which the transaction T2

updates the record. Now when the transaction T1 reads the record then the new value will be inconsistent with the previous value.

Transaction-X	Time	Transaction-Y
——	t1	——
Read Balance of Acc-1 sum <-- 200 Read Balance of Acc-2	t2	——
Sum <-- Sum + 250 = 450	t3	——
——	t4	Read Balance of Acc-3
——	t5	Update Balance of Acc-3 150 --> 150 - 50 --> 100
——	t6	Read Balance of Acc-1
——	t7	Update Balance of Acc-1 200 --> 200 + 50 --> 250
Read Balance of Acc-3	t8	COMMIT
Sum <-- Sum + 250 = 450	t9	——

Transaction-X is doing sum of all balance while, Transaction-Y is transferring an amount 50 from Account-1 to Account-3.

- Here, transaction-X produces the result of 550 which is incorrect. If we write this produced result in the database then database will become in inconsistent state because the actual sum is 600.
- Here, transaction-X has seen an inconsistent state of the database.

Concurrency Control Protocol

Concurrency control protocols ensure atomicity, isolation, and serializability of concurrent transactions. The concurrency control protocol can be divided into three categories:

1. Lock based protocol
2. Time-stamp protocol
3. Validation based protocol

INDEXING

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.
Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

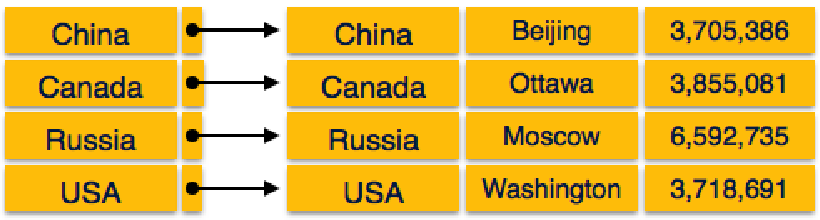
- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

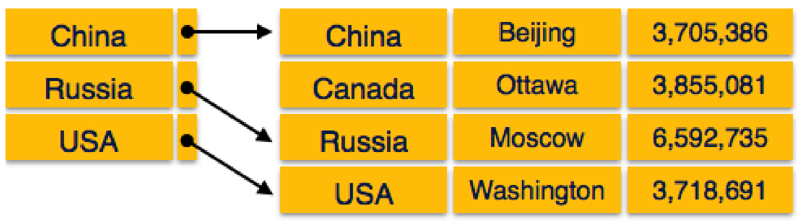
Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.


China
Canada
Russia
USA
China | Beijing | 3,705,386
Canada | Ottawa | 3,855,081
Russia | Moscow | 6,592,735
USA | Washington | 3,718,691

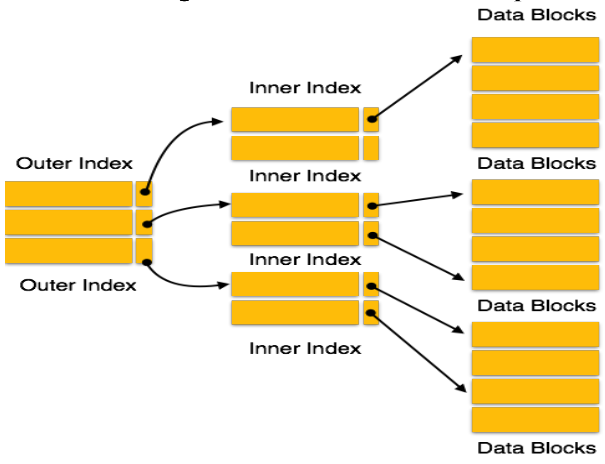
Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.


China
Russia
USA
China | Beijing | 3,705,386
Canada | Ottawa | 3,855,081
Russia | Moscow | 6,592,735
USA | Washington | 3,718,691

Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.

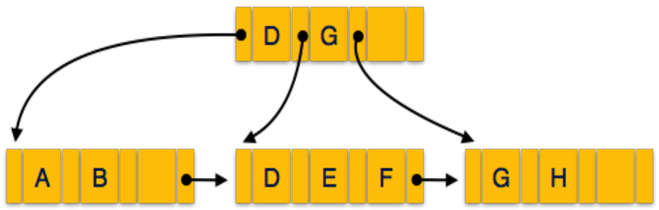


Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B+ Tree

A B+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B+ tree denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height, thus balanced.

Additionally, the leaf nodes are linked using a link list; therefore, a B+ tree can support random access as well as sequential access.
Structure of B+ Tree
Every leaf node is at equal distance from the root node. A B+tree is of the order **n** where **n** is fixed for every B+ tree.



Internal nodes −

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes −

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.