

Database Management System

Database Management System

A database management system is a collection of interrelated data and a set of programs to access those data. It manages new large amount of data and supports efficient access to new large amounts of data.

Data

Known facts that can be stored or recorded is called data.

It can be of two types

Persistent Data It continues to exist even, when the system is not active.

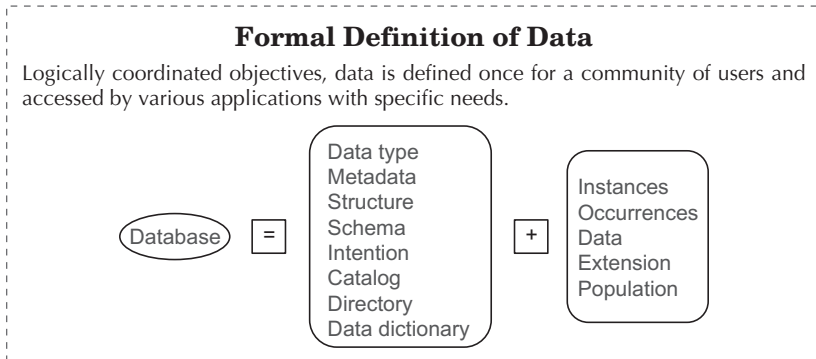
Transient Data It created while an application is running and not needed, when the application has terminated. It must be stored in secondary memory.

Database

Database is a collection of related data with

- A logically coherent structure (can be characterised as a whole).
- Some inherent meaning (represents some partial view of a portion of the real world).
- Some specific purpose and for an intended group of users and applications.
- A largely varying size (from a personal phone book directory to the phone book directory of a city or state).

- A scope or content of varying breadth (from a personal list of addresses to a multimedia encyclopedia).
- A physical organisation of varying complexity (from a manual personal list, managed with simple files, to huge multiuser databases with geographically distributed data and access).



The information contained in a database is represented on two levels

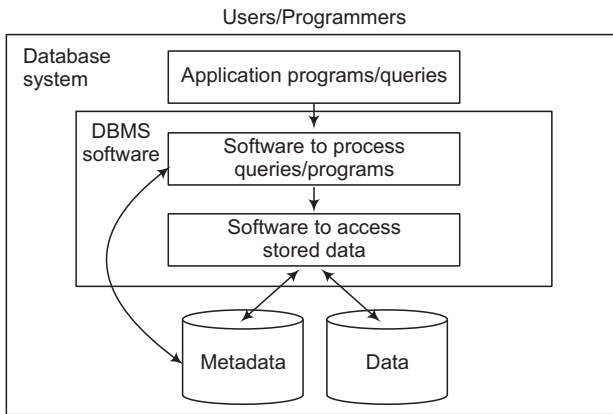
- (i) Data (which is large and is being frequently modified)
- (ii) Structure of data (which is small and stable in time)

Basic Concept of DBMS

It is a collection of general purpose, application independent programs providing services to

- Define the structure of a database, *i.e.*, data types and constraints that the data will have to satisfy.
- Manage the storage of data, safely for long periods of time, on some storage medium controlled by the DBMS.
- Manipulate a database, with efficient user interfaces to query the database to retrieve specific data, update the database to reflect changes in the world, generate reports from the data.
- Manage database usage for users with their access rights, performance optimisation, sharing of data among several users, security from accidents or unauthorised use.
- Monitor and analyse database usage.

Database Management System (DBMS) provides efficient, reliable, convenient and safe multiuser storage of and access to massive amounts of persistent data.



Architecture of database and DBMS software

Key People Involved in a DBMS

The followings are the key people involved in a DBMS as

DBMS Implementer

Person who builds system.

Database Designer

Person responsible for preparing external schemas for applications ,identifying and integrating user needs into a conceptual (or community or enterprise) schema.

Database Application Developer

Person responsible for implementing database application programs that facilitate data access for end users.

Database Administrator

Person responsible for define the internal schema, sub-schemas (with database designers) and specifying mappings between schemas, monitoring database usage and supervising DBMS functionality (e.g., access control, performance optimisation, backup and recovery policies, conflict management).

End Users

There are two categories of end users

- **Casual users** Occasional unanticipated access to database. e.g., tourists, managers.

- **Parametric users** Users who query and update the database through fixed programs (invoked by non-programmer users) e.g., banking.

Important Functions on a Database

- **Structure Definition** Declare files or relations (*i.e.*, tables + data types). e.g., Employee (Emp_Name, Emp_No, Salary, Dept_ID)
- **Dept** (Dept_Name, Dept_ID)
- **Population** Input data about specific employee, department.
- **Querying** List name of employees who are getting salary more than 3 lakh in department marketing.
- **Reporting** We can prepare a report having employee names with their department name using join on employee table and dept table.
- **Modification and Update of Population** We can create a new department name for the table department and also we can update salary of a particular employee.
- **Modification of Structure and Schema** We can create a new relation (*i.e.*, table) for '(Emp_No, Skills).' We can add address attribute to relation employee.

Instances and Schemas and Data Model

Data Model It provides mechanisms (languages) for defining data structures and operations for retrieval and modification of data.

Schema/Intension The overall design of the database or description of data in terms of a data model. Schemas do not change frequently. In addition to data structures, the schema also comprises the definition of domains for data elements (attributes) and the specification of constraints, to define the data structure part of the schema. While intension is a constant value that gives the name, structure of table and the constraints laid on it.

Instance/Extension

Databases change over time because records are inserted and deleted frequently. The collection of information stored in the database at a particular moment is called an instance of the database. While extension means the number of tuples present in a table at any instance. This is dependent on time.

Data Abstraction

Hiding the complexity from users through several levels of abstraction, to simplify users' interactions with the system, as many database systems users are not computer trained.

Levels of Data Abstraction

There are three levels of data abstraction as given below

Physical Level

It is lowest level of abstraction and describes how the data are actually stored and complex low level data structures in detail. At the physical level, an employee record can be described as a block of consecutive storage locations. e.g., words or bytes.

Logical Level

It is the next higher level of abstraction and describes what data are stored and what relationships exist among those data. At the logical level, each such record is described by a type definition and the interrelationship of these record types is defined as well. Database administrators usually work at this level of abstraction.

View Level

It is the highest level of abstraction and describes only part of the entire database and hides the details of the logical level.

- At the view level, several views of the database are defined and database users see these views.
- The views also provide a security mechanism to prevent users from accessing certain parts of the database. e.g., tellers in a bank can see information on customer accounts but they cannot access information about salaries of employees.

Key Points

- ♦ Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
 - ♦ Many users of the database system do not need all stored information.
 - ♦ Users classified need to access only a part of the database.
 - ♦ The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.
-

Schema

A schema is also known as database schema. It is a logical design of the database and a database instance is a snapshot of the data in the database at a given instant of time. A **relational schema** consists of a list of attributes and their corresponding domains.

Types of Schemas

It can be classified into three parts, according to the levels of abstraction

Physical/Internal Schema Describes the database design at the physical level.

Logical/Conceptual Schema/Community User View Describes the database design at the logical level.

Sub-schemas/View/External Schema Describes different views of the database, views may be queried, combined in queries with base relations, used to define other views in general, not updated freely.

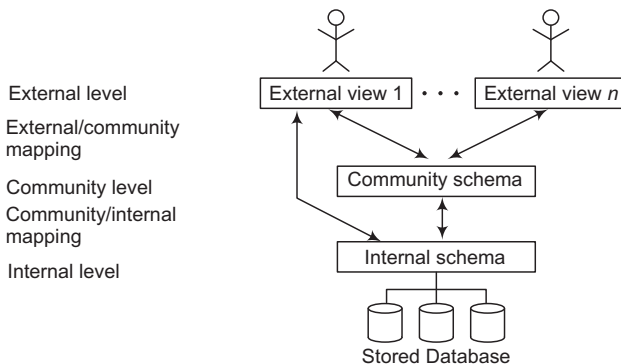
Schema Architecture

Data Independence

Possibility to change the schema at one level without having to change it at the next higher level (nor having to change programs that access it at that higher level).

There are two parts of data independence

Logical Data Independence Refers to the immunity of the external schema to changes in the conceptual schema (i.e., community schema). e.g., add new record or field.



Architecture of schema

Physical Data Independence Refers to the immunity of the conceptual schema to changes in the internal schema. e.g., addition of an index should not affect existing one.

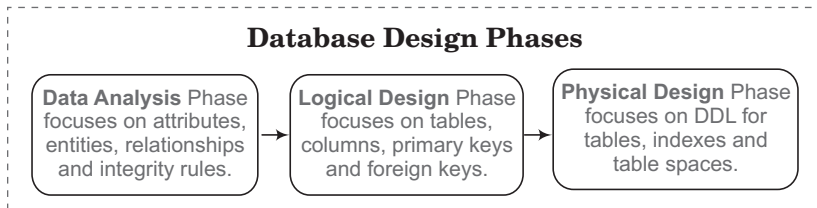
Database Functions and Application Functions

Application Program Functions

To be programmed in application programs.

Database Functions or DBMS Functions

Supplied by the DBMS and invoked in application programs.



Transaction

One execution of a user program (executing the same programs several times corresponds to several transactions). Basic unit of change as seen by the DBMS.

OLTP (On-Line Transaction Processing) applications (e.g., banking and airline systems) with multiple simultaneous users.

Functionality of DBMS

- | | |
|--------------------------------------|------------------------|
| 1. Concurrency control | 2. Backup and recovery |
| 3. Redundancy management | 4. Access control |
| 5. Performance optimisation | 6. Metadata management |
| 7. Active features (rules, triggers) | |

Concurrency Control

It is responsible for ensuring correctness of competing accesses to same data. One or more Structure Query Language (SQL) statements altogether treated as one single unit.

Correctness of data requires four desirable properties (ACID properties) e.g., concurrency control means there should not be two simultaneous withdrawals from the same bank account or there should not be multiple reservations of the same airplane seat.

Backup and Recovery

- Facilities for recovering from hardware and software failures.
- If the computer system fails during a complex update program, the database must be restored to its state before the program started, or the program must be resumed, where it was interrupted so that its full effect is recorded in the database.
- In a multiuser environment, it is more complex and important.

Redundancy Management

Redundancy means storing several copies of the same data. Redundant entries are frequent in traditional file processing; a goal of the database approach was to control redundancy as much as possible.

Problems with redundancy includes waste of storage space, duplication of effort to perform a single conceptual update, danger of introducing inconsistency, if multiple updates are not coordinated.

Access Control

Responsible for enforcing security and authorisation (e.g., who can create new bank accounts) and data (e.g., which bank accounts can I see).

It is all about who accesses what data, to do what, when, from where etc.

Some examples of access privileges are as follows

- To create a database
- To authorise (grant) additional users to access the database, access some relations, create new relations and update the database.
- To revoke privileges.

Key Points

- ♦ In a multiuser database, access control is mandatory e.g., for confidentiality.
 - ♦ Various ways to access data (e.g., read only, read and update).
 - ♦ The data dictionary holds information about users and their access privileges (e.g., name and password).
-

Performance Optimisation

Performing physical reorganisations to enhance performance e.g., adding index, dropping index, sorting file.

Performance optimisation is made possible by physical data independence and high level data models with user programs which can be optimised through DBMS software.

Metadata Management

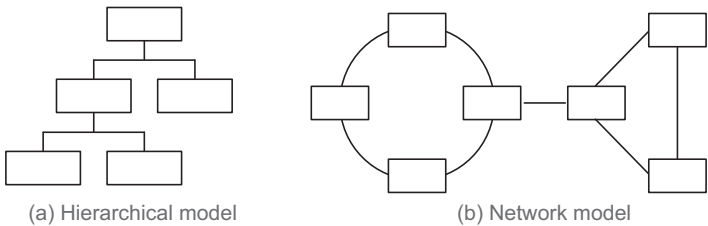
Metadata means data about data. Metadata is maintained in a special database, which we can call system catalog or data dictionary. It involves storing of information about other information. With different types of media being used, references to the location of the data can allow management of diverse repositories.

Active Features

Data objects, database statistics, physical structures and access paths, user access privileges etc, are active features of DBMS.

Types of Database Model

The database model can be of three types as given below



Attributes/Column

Name	Roll	Class
Ram	001	CS
Shyam	002	IT

Value (c) Relational model

Database models

Relational Database Management System (RDBMS)

A system in which users access data with use of relation (*i.e.*, data must be available in tabular form *i.e.*, as a collection of tables, where each table consisting a set of rows and columns). That provides a variety of relational operator so that we can manipulate the data in tabular form.

Features of RDBMS

There are many features of RDBMS

- We can create multiple relations (tables) and feed data into them.
- Provides an interactive query language.
- We can retrieve information from more than one table using join concept.
- Provides a catalog or dictionary which consists of system tables.

RDBMS Vocabulary

Relation A table.

Attribute A column in a table.

Degree Number of attributes in a relation.

Cardinality Number of tuples in a table.

Domain or Type A pool of values from which specific attributes draw their values.

NULL Special value for 'unknown' or 'undefined'.

Relational Data Model It provides mechanisms (languages) for defining data structures, operations for retrieval and modification of data and integrity constraints.

Keys

An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a **key** for that entity set.

Different types of keys are as follows

Super Key

It is a set of one or more attributes that allow us to identify uniquely an entity in the entity set. *e.g.*, the Customer_Id attribute of the entity set, customer is sufficient to distinguish one customer entity from another. Thus, Customer_Id is a super key. Similarly, the combination of Customer_Name and Customer_Id is a super key for the entity set customer. The Customer_Name attribute of customer is not a super key, because several people might have the same name.

Candidate Key

Since, as we saw a super key may contain extraneous attributes (*i.e.*, Customer_Name here in the above case). If K is a super key, then candidate key is any superset of K . We are often interested in super keys for which no proper subset is a super key. Such minimal super keys are called candidate keys. Suppose that a combination of Customer_Name and Customer_Street is sufficient to distinguish among members of the customer entity set. Then, both {Customer_Id} and {Customer_Name, Customer_Street} are candidate keys. Although, the attributes Customer_Id and Customer_Name together can distinguish customer entities, their combination does not form a candidate key, since the attribute Customer_Id alone is a candidate key.

Primary Key

A candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate and super) is a property of the entity set rather than of the individual entities.

Alternate Key

A table may have one or more choices for the primary key. Collectively these are known as candidate keys as discuss earlier. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.

Secondary Key

An attribute or set of attributes that may not be a candidate key but that classifies the entity set on a particular characteristics. *e.g.*, suppose entity set employee having the attribute department whose value identifies all instances employee who belong to a given department.

Foreign Key

A foreign key is generally a primary key from one table that appears as a field in another, where the first table has a relationship to the second. In other words, if we had a table TBL1 with a primary key. A that linked to a table TBL2, where A was a field in TBL2, then A would be a foreign key in TBL2.

Simple and Composite Key

Any key consisting of a single attribute is called a simple key, while that consisting of a combination of attributes is called a composite key.

E-R Modeling

Entity-Relationship model (ER model) in software engineering is an abstract way to describe a database. Describing a database usually starts with a relational database, which stores data in tables. Some of the data in these tables point to data in other tables for instance, your entry in the database could point to several entries for each of the phone numbers that are yours. The ER model would say that you are an entity, and each phone number is an entity, and the relationship between you and the phone numbers is 'has a phone number'.

Relationship

An association among entities. e.g., There is a relationship between employees and department, which can be named as Works_in.

Relationship Set

An association of entity sets e.g., Employee_Department

- A relationship instance is an association of entity instances e.g., Shyam_Sales.
- Same entity set could participate in different relationship sets.
- An n -array relationship set R relates n entity sets.
- A relationship set involving three entity sets, is known as a ternary relationship.

Entity

Anything that exists and can be distinguished/ real world object which can be distinguished from other objects. e.g., student.

Entity Set

A group of similar entities, e.g., all students.

- All entities in an entity set have the same set of attributes.
- Each entity set has a key.
- Can be mapped to a relation easily.

Weak Entity Set and Strong Entity Set

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. An entity set that has a primary key is termed a strong entity set. For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated with an identifying entity i.e., the weak entity set is said to be existence dependent on the identifying entity set.

Attribute

Properties that describe an entity or we can say attributes are descriptive properties possessed by each member of an entity set and each attribute has a domain.

There are many types of attributes

Composite Attribute

Attributes which can have component attribute. e.g., a composite attribute name, with component attributes First_Name, Middle_Name and Last_Name.

Derived Attribute

The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the customer entity set has an attribute Loans_Held, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of loan entities associated with that customer.

Descriptive Attribute

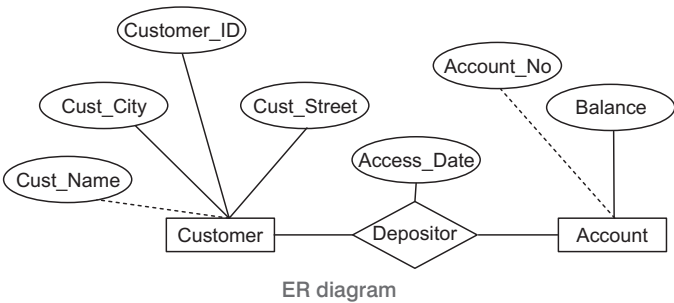
If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. e.g., consider a relationship set depositor with entity sets customer and account. We could associate the attribute Access_Date to that relationship to specify the most recent date on which a customer accessed an account.

Single Valued Attribute












Attribute which has only one value, e.g., the Employee_Number attribute for a specific Employee_entity refers to only one employee number.

Multi Valued Attribute

Attributes which can have 0, 1 or more than 1 values. An employee entity set with the attribute Phone_Number. An employee may have zero, one or several phone numbers and different employees may have different numbers of phones.



Notations/Shapes in ER Modeling

Rectangle represents entity type.	
Double/Bold rectangle represent weak entity type.	 or 
Diamond represents relationship type.	
Double/Bold diamond represents weak relationship type.	 or 
Ellipse represents attribute type.	
Double ellipse represents multivalued attribute.	
Dashed ellipse denotes derived attribute.	
Line a link attribute to entity sets and entity sets to relationship sets	
Double lines which indicate total participation of an entity in a relationship set <i>i.e.</i> , each entity in the entity set occurs in atleast one relationship in that relationship set.	

Mapping Cardinalities/Cardinality Ratio/Types of Relationship

Expresses the number of entities to which another entity can be associated *via* a relationship set. For a binary relationship set R between entity sets A and B , the mapping cardinality must be one of the following

One to One

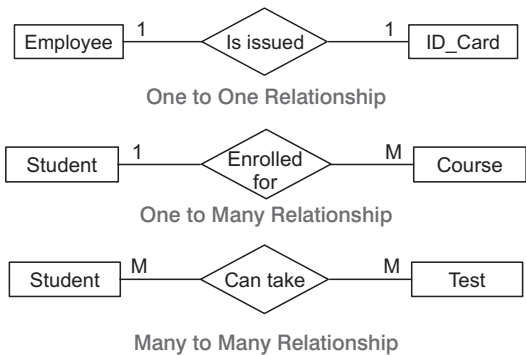
An entity in A is associated with at most one entity in B and an entity in B is associated with at most one entity in A .

One to Many

An entity in A is associated with any number (zero or more) of entities in B . An entity in B , however, can be associated with at most one entity in A .

Many to Many

An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.



Extended E-R Features

The extended E-R features are given below

Specialization

Consider an entity set person with attributes name, street and city. A person may be further classified as one of the following

- (i) Customer
- (ii) Employee

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes. The process of designating subgroupings within an entity set is called specialization.

The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicitly.

Generalization

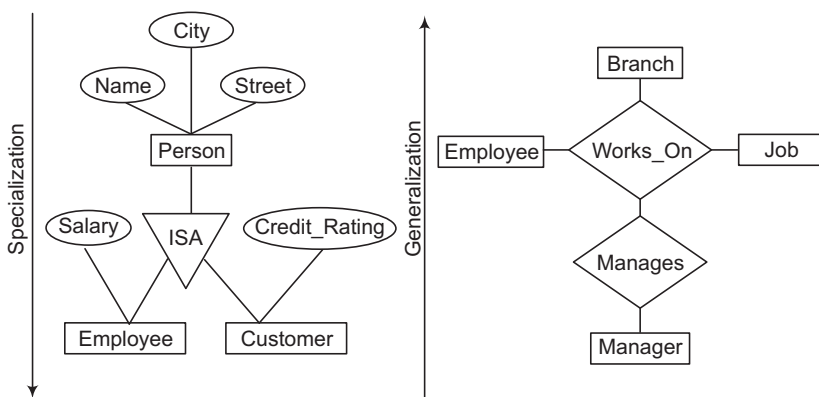
Basically generalization is a simple inversion of specialization. Some common attributes of multiple entity sets are chosen to create higher level entity set. If the customer entity set and the employee entity set are having several attributes in common, then this commonality can be expressed by generalization.

Here, person is the higher level entity set and customer and employee are lower level entity sets. Higher and lower level entity sets also may be designated by the terms super class and subclass, respectively. The person entity set is the super class of the customer and employee subclasses.

Aggregation

Aggregation is used when we have to model a relationship involving entity set and a relationship set. Aggregation is an abstraction through which relationships are treated as higher level entities.

Suppose the relationship set Works_On (relating the entity sets employee, branch and job) as a higher_level entity set called Works_On. Such an entity set is treated in the same manner as is any other entity set. We can create a binary relationship manages between Works_On and manager to represent who manages what tasks. Aggregation is meant to represent a relationship between a whole object and its component parts.



Example of generalization and specialization Example of aggregation

Integrity Constraints

Necessary conditions to be satisfied by the data values in the relational instances so that the set of data values constitute a meaningful database.

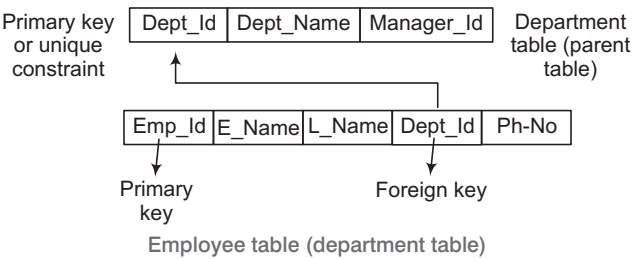
There are four types of integrity constraints

Domain Constraint The value of attribute must be within the domain.

Key Constraint Every relation must have a primary key.

Entity Integrity Constraint Primary key of a relation should not contain NULL values.

Referential Integrity Constraint In relational model, two relations are related to each other over the basis of attributes. Every value of referencing attributes must be NULL or be available in the referenced attribute.



Relational Algebra and Relational Calculus

Relational model is completely based on relational algebra. It consists of a collection of operators that operate on relations.

Its main objective is data retrieval. It is more operational and very much useful to represent execution plans, while relational calculus is non-operational and declarative. Here, declarative means user define queries in terms of what they want, not in terms of how compute it.

Relational Query Languages

Used for data manipulation and data retrieval. Relational model support simple yet powerful query languages. To understand SQL, we need good understanding of two relational query language (*i.e.*, relational algebra and relational calculus).

Basic Operation in Relational Algebra

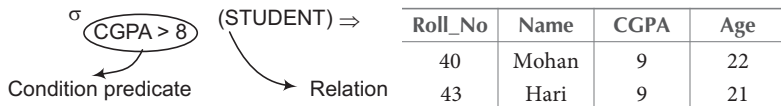
The operations in relational algebra are classified as follows

Selection

The select operation selects tuples/rows that satisfy a given predicate or condition. We use (σ) to denote selection. The predicate/condition appears as a subscript to σ .

e.g., Consider the below relation STUDENT

Roll_No	Name	CGPA	Age
30	Ram	7	20
35	Shyam	8	21
40	Mohan	9	22
43	Hari	9	21



Projection

It selects only required/specified columns/attributes from a given relation/table. Projection operator eliminates duplicates (*i.e.*, duplicate rows from the result relation) e.g., consider the STUDENT relation.

$\Pi_{\text{Age}}(\text{STUDENT}) \Rightarrow$

Age
20
21
22

$\Pi_{\text{Name, CGPA}}(\text{STUDENT}) \Rightarrow$

Name	CGPA
Ram	7
Shyam	8
Mohan	9
Hari	9

Combining selection and projection

$\Pi_{\text{Name, CGPA}}(\sigma_{\text{CGPA} > 8}(\text{STUDENT}))$
 \Downarrow

Name	CGPA
Mohan	9
Hari	9

Union

It forms a relation from rows/tuples which are appearing in either or both of the specified relations. For a union operation $R \cup S$ to be valid, *below two conditions must be satisfied*.

- The relations R and S must be of the same entity. *i.e.*, they must have the same number of attributes.
- The domains of the i th attribute of R and i th attribute of S must be the same, for all i .

Intersection

It forms a relation of rows/ tuples which are present in both the relations R and S . As with the union operation, we must ensure that both relations are compatible.

	<table><tr><th>Name</th><th>Age</th></tr><tr><td>Ram</td><td>20</td></tr><tr><td>Mohan</td><td>21</td></tr></table>	Name	Age	Ram	20	Mohan	21	<table><tr><th>Name</th><th>Age</th></tr><tr><td>Ram</td><td>20</td></tr><tr><td>Shyam</td><td>22</td></tr></table>	Name	Age	Ram	20	Shyam	22
Name	Age													
Ram	20													
Mohan	21													
Name	Age													
Ram	20													
Shyam	22													
	R	S												
$R \cup S \Rightarrow$	<table><tr><th>Name</th><th>Age</th></tr><tr><td>Ram</td><td>20</td></tr><tr><td>Shyam</td><td>22</td></tr><tr><td>Mohan</td><td>21</td></tr></table>	Name	Age	Ram	20	Shyam	22	Mohan	21	$R \cap S \Rightarrow$				
Name	Age													
Ram	20													
Shyam	22													
Mohan	21													
(Union operation)		(Intersection operation)												
		<table><tr><th>Name</th><th>Age</th></tr><tr><td>Ram</td><td>20</td></tr></table>	Name	Age	Ram	20								
Name	Age													
Ram	20													

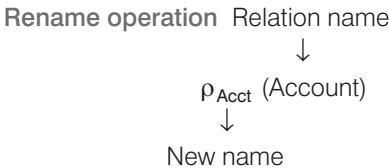
Set Difference

It allows us to find tuples that are in one relation but are not in another. The expression $R - S$ produces a relation containing those tuples in R but not in S .

$R - S \Rightarrow$	<table><tr><th>Name</th><th>Age</th></tr><tr><td>Mohan</td><td>21</td></tr></table>	Name	Age	Mohan	21	$(\Pi_{\text{Name, Age}} (R) - \Pi_{\text{Name, Age}} (S))$
	Name	Age				
Mohan	21					

Cross Product/Cartesian Product

Assume that we have n_1 tuples in R and n_2 tuples in S . Then, there are $n_1 * n_2$ ways of choosing a pair of tuples; one tuple from each relation. So, there will be $(n_1 * n_2)$ tuples in result relation P if $P = R \times S$.



$\rho_x(E_1)$, where x is the new name for the result of E_1 and E_1 may be an expression.

Schema Refinement/Normalization Decomposition of complex records into simple records. Normalization reduces redundancy using non-loss decomposition principle.

Shortcomings of Redundancy Data redundancy causes three types of anomalies insert, update and delete. Storage problem (*i.e.*, wastage of storage).

Decomposition Splitting a relation R into two or more subrelation R_1 and R_2 . A fully normalized relation must have a primary key and a set of attributes.

Database Design Goal (Schema Refinement)

There are many goals for the design of a database. *Here are some of them listed*

The Database is Compressive It includes all the needed data and connections.

The Database is Understandable There is a clear structure which leads to easy, flexible and fast reading and updating of the data.

The Database is Expandable It is possible to change the structure of the database with a minimum change to the existing software.

- 0% redundancy

Decomposition should satisfy

- (i) Lossless join
- (ii) Dependency preservice

Lossless Join Decomposition

Join between the subrelations should not create any additional tuples or there should not be a case such that more number of tuples in R_1 than R_2

$$R \subset R_1 \bowtie R_2 \Rightarrow (\text{Lossy})$$

$$R \equiv R_1 \bowtie R_2 \Rightarrow (\text{Lossless})$$

Dependency Preservice

Because of decomposition, there must not be loss of any single dependency.

Functional Dependency (FD)

Dependency between the attribute is known as functional dependency. Let R be the relational schema and X, Y be the non-empty sets of attributes and t_1, t_2, \dots, t_n are the tuples of relation R .

$X \rightarrow Y$ {values for X functionally determine values for Y }

If the above condition holds, it means if

$t_1 \cdot X = t_2 \cdot X$, then $t_1 \cdot Y = t_2 \cdot Y$

tup Les	X	Y
t_1	a	b
t_2	a	b

Relation R_1 , which holds $X \rightarrow Y$

tup Les	X	Y
t_3	a	b
t_4	a	c

Relation R_2 , which does not hold $X \rightarrow Y$

Trivial Functional Dependency

If $X \supseteq Y$, then $X \rightarrow Y$ will be trivial FD.

Here, X and Y are set of attributes of a relation R .

In trivial FD, there must be a common attribute at both the sides of ' \rightarrow ' arrow.

$$\left. \begin{array}{l} S_{id} S_{name} \rightarrow S_{name} \\ S_{id} S_{name} \rightarrow S_{id} \end{array} \right\} \text{trivial FD}$$

Non-Trivial Functional Dependency

If $X \cap Y = \emptyset$ (no common attributes) and $X \rightarrow Y$ satisfies FD, then it will be a non-trivial FD.

$$\left. \begin{array}{l} C_{id} \rightarrow C_{name} \\ S_{id} \rightarrow S_{name} \end{array} \right\} \text{non-trivial FD}$$

(no common attribute at either side of ' \rightarrow ' arrow)

Case of semi-trivial FD

$$S_{id} \rightarrow S_{id} S_{name} \text{ (semi-trivial)}$$

Because on decomposition, we will get

$$S_{id} \rightarrow S_{id} \text{ (trivial FD) and}$$

$$S_{id} \rightarrow S_{name} \text{ (non-trivial FD)}$$

Properties of Functional Dependence (FD)

- **Reflexivity** If $X \supseteq Y$, then $X \rightarrow Y$ (trivial)
- **Transitivity** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- **Augmentation** If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Splitting or Decomposition** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Union** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Attribute Closure

Suppose $R(X, Y, Z)$ be a relation having set of attributes *i.e.*, (X, Y, Z) , then (X^+) will be an attribute closure which functionally determines other attributes of the relation (if not all then atleast itself).

DBMS versus RDBMS

DBMS	RDBMS
<ul style="list-style-type: none"> • In DBMS, relationship between two tables or files are maintained programmatically, morganatically 	In RDBMS, relationship between two tables or files can be specified at the time of table creation
<ul style="list-style-type: none"> • DBMS does not support client/server architecture 	Most of the RDBMS supports client/server architecture
<ul style="list-style-type: none"> • DBMS does not support distributed database 	Most of the RDBMS support distributed databases
<ul style="list-style-type: none"> • In DBMS, there is no security of data 	In RDBMS, <i>there are multiple levels of security</i>
	(i) Cogging in at o/s Level
	(ii) Comand Level
	(iii) Object level
<ul style="list-style-type: none"> • Each table is given an extension in DBMS 	Many tables are grouped in one database in RDBMS

Normal Forms/Normalization

In relational database design, the normalization is the process for organizing data to minimize redundancy. Normalization usually involves dividing a database into two or more tables and defining relationship between the tables.

The normal forms define the status of the relation about the individuated attributes. *There are five types of normal forms*

First Normal Form (1NF)

Relation should not contain any multivalued attributes or relation should contain atomic attributes.

Anomalies in Database

Stud_Id	Stud_Name	Course_Name
S ₁	A	C/C++
S ₂	B	C++
S ₃	B	C++/Java

(STUDENT) Relation

The above relation is not in 1 NF



Stud_Id	Stud_Name	Course_Name
S ₁	A	C
S ₁	A	C++
S ₂	B	C++
S ₃	B	C++
S ₃	B	Java

The above relation is in 1NF but now Stud_Id is no more a primary key.

Now in the above case, we need to modify our primary key which is (Stud_Id, Course-Name)

Note The main disadvantage of 1NF is high redundancy.

Second Normal Form (2NF)

Relation R is in 2NF if and only if

- R should be in 1NF.
- R should not contain any partial dependency.

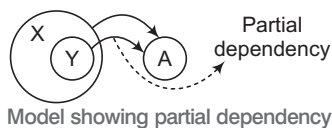
Partial Dependency

Let R be the relational schema having X, Y, A , which are non-empty set of attributes, where

X = Any candidate key of the relation.

Y = Proper subset of any candidate key

A = Non-prime attribute (*i.e.*, A doesn't belong to any candidate key)



In the above example, $X \rightarrow A$ already exists and if $Y \rightarrow A$ will exist, then it will become a partial dependency, if and only if

- Y is a proper subset of candidate key.
- A should be non-prime attribute.

If any of the above two conditions fail, then $Y \rightarrow A$ will also become fully functional dependency.

Full Functional Dependency

A functional dependency $P \rightarrow Q$ is said to be fully functional dependency, if removal of any attribute S from P means that the dependency doesn't hold any more.

(Student_Name, College_Name \rightarrow College_Address)

Suppose, the above functional dependency is a full functional dependency, *then we must ensure that there are no FDs as below.*

(Student_Name \rightarrow College_Address)

or (College_Name \rightarrow Collage_Address)

Third Normal Form (3NF)

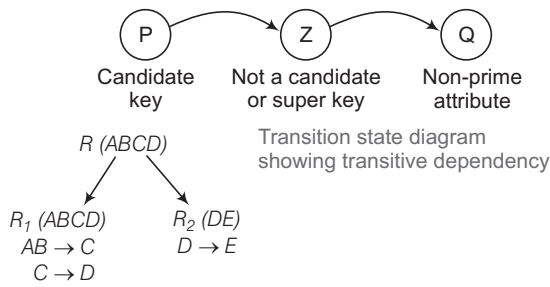
Let R be a relational schema, then any non-trivial FD $X \rightarrow Y$ over R is in 3NF, if

- X should be a candidate key or super key.
- or
- Y should be a prime attribute.
- Either both of the above conditions should be true or one of them should be true.
- R should not contain any transitive dependency.
- For a relation schema R to be a 3NF, it is necessary to be in 2NF.

Transitive Dependency

A FD, $P \rightarrow Q$ in a relation schema R is a transitive if

- There is a set of attributes Z that is not a subset of any key of R .
- Both $X \rightarrow Z$ and $Z \rightarrow Y$ hold



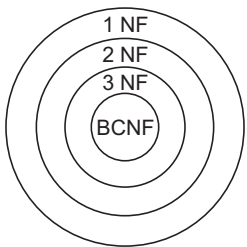
- The above relation is in 2NF.
- In relation R_1 , C is not a candidate key and D is non-prime attribute. Due to this, R_1 fails to satisfy 3NF condition. Transitive dependency is present here.

$AB \rightarrow C$ and $C \rightarrow D$, then $AB \rightarrow D$ will be transitive.

Boycee Codd Normal Form (BCNF)

Boycee Codd Normal Form (BCNF) Let R be the relation schema and $X \rightarrow Y$ be the any non-trivial FD over R is in BCNF if and only if X is the candidate key or super key.

$X \rightarrow Y$
candidate / super key } If R satisfies this dependency, then of course it satisfy 2NF and 3NF.



Hierarchy of normal forms

Summary of 1 NF, 2 NF and 3 NF

Normal Form	Test	Remedy (Normalization)
1 NF	Relation should have no non-atomic attributes or nested relations.	Form name relation for each non-atomic attribute or nested relation.
2 NF	For relations where primary key contains multiple attributes, no non-key attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attributes. Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
3 NF	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes), <i>i.e.</i> , there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and setup a relation that includes the non-key attribute(s) that functionally determine(s) other non-key attribute(s)

Fourth Normal Form (4NF)

4NF is mainly concerned with multivalued dependency. A relation is in 4NF if and only if for every one of its non-trivial multivalued dependencies $X \twoheadrightarrow Y$, X is a super key (*i.e.*, X is either a candidate key or a superset).

Fifth Normal Form (5NF)

It is also known as Project Join Normal Form (PJ/NF). 5NF reduces redundancy in relational database recording multivalued facts by isolating semantically related multiple relationships.

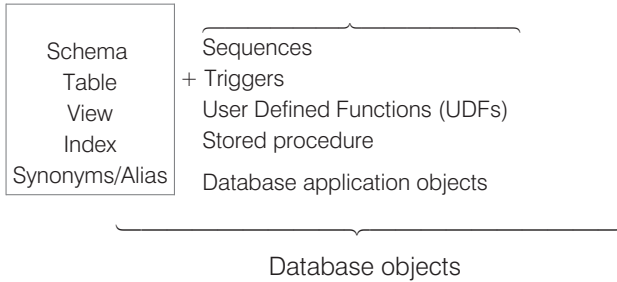
A table or relation is said to be in the 5NF, if and only if every join dependency in it, is implied by the candidate keys.

Key Points

- The Normal Forms (NF) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies
- Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP)
- OLTP applications are characterized by a high volume of small transactions such as updating a sales record at a supermarket checkout counter.
- The expectation is that each transaction will leave the database in a consistent state

SQL

Structured Query Language (SQL) is a language that provides an interface to relation database systems. SQL was developed by IBM in the 1970, for use in system R and is a defacto standard, as well as an ISO and ANSI standard.



- To deal with the above database objects, we need a programming language and that programming language is known as SQL.

Three subordinate languages of SQL are

Data Definition Language (DDL)

It includes the commands as

- **CREATE** To create tables in the database.
- **ALTER** To modify the existing table structure.
- **DROP** To drop the table with table structure.
- **Data Manipulation Language (DML)** It is used to insert, delete, update data and perform queries on these tables. *Some of the DML commands are given below.*
- **INSERT** To insert data into the table.
- **SELECT** To retrieve data from the table.
- **UPDATE** To update existing data in the table.
- **DELETE** To delete data from the table.

Data Control Language (DCL)

It is used to control user's access to the database objects. *Some of the DCL commands are*

- **GRANT** Used to grant select/insert/delete access.
- **REVOKE** Used to revoke the provided access.

Transaction Control Language (TCL)

It is used to manage changes affecting the data.

- **COMMIT** To save the work done, such as inserting or updating or deleting data to/from the table.
- **ROLLBACK** To restore database to the original state, since last commit.
- **SQL Data Types** SQL data types specify the type, size and format of data/information that can be stored in columns and variables.

Key Points

- ♦ SQL is a special-purpose programming language designed for managing data held in a Relational Database Management Systems (RDBMS).
 - ♦ SQL based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language.
-

Various Data Types in SQL

- Data Time time-stamp
- Char Big int Integer
- Decimal Small int Double

There are so many other data types also.

Database Constraints

These are user defined that let us restrict the behaviours of column. We can create constraints when we define a table with a SQL CREATE statement.

Inline Constraint

A constraint defined on the same line as its column.

Out of Line Constraint

A constraint defined on it's own line in a CREATE statement. This type of constraint must reference the column that they constrain.

Constraint Types with Description

Name of constraint	Table level/column level/Row level/External level	Description
NOT NULL	Column level	Restricts a column by making it mandatory to have some value.
Unique	Table level	Checks whether a column value will be unique among all rows in a table.
Primary key	Column level and Table level	Checks whether a column value will be unique among all rows in a table and disallows NULL values.
Check constraint	Column level Row level External level	Restrict a column value to a set of values defined by the constraint.
Foreign key constraint	Column level External level	Restrict the values that are acceptable in a column or group of columns of a table to those values found in a listing of the column/group of columns used to define the primary key in other table.

Key Points

- ♦ The most common operation in SQL is the query, which is performed with the declarative SELECT statement.
- ♦ SELECT retrieves data from one or more tables, or expressions.
- ♦ Standard SELECT statements have no persistent effects on the database.
- ♦ Queries allow the user to describe desired data, leaving the Database Management System (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

Default Constraint

It is used to insert a default value into a column, if no other value is specified at the time of insertion.

Syntax

```
CREATE TABLE Employee
{
    Emp_id int NOT NULL,
    Last_Name varchar (250),
    City varchar (50)DEFAULT 'BANGALURU'
}
```

DDL Commands

```
1. CREATE TABLE < Table_Name>
{
    Column_name 1 < data_type >,
    Column_name 2 < data_type >
}
```

2. ALTER TABLE < Table_Name >
ALTER Column < Column_Name> SET NOT NULL
3. RENAME < object_type >object_name >to <new_name >
4. DROP TABLE <Table_Name>

DML Commands

SELECT $A_1, A_2, A_3, \dots, A_n$ what to return
FROM $R_1, R_2, R_3, \dots, R_m$ relations or table

WHERE condition filter condition *i.e.*, on what basis, we want to restrict the outcome/result.

If we want to write the above SQL script in the form of relational calculus, we use the following syntax

$$\{\Pi_{A_1 \dots A_n} (\sigma_{\text{condition}} (R_1 \times R_2 \times \dots \times R_m))\}$$

Comparison operators which we can use in filter condition are
(=, >, <, >=, <=, <>) '<>' means not equal to.

INSERT Statement

Used to add row (s) to the tables in a database.

INSERT INTO Employee (F_Name, L_Name)VALUES ('Atal', 'Bihari')

Key Points

- * Data values can be added either in every column in a row or in same columns in a row by specifying the columns and their data.
- * All the columns that are not listed in the column list in INSERT statement, will receive NULL.

UPDATE Statement

It is used to modify/update or change existing data in single row, group of rows or all the rows in a table.

e.g.,

```
UPDATE Employee
SET City = 'LUCKNOW'
WHERE Emp_Id BETWEEN
9 AND 15;
```

An example of selective update which will update some rows in a table.

```
UPDATE Employee SET City
= 'LUCKNOW' ;
```

Example of global update which will update city column for all the rows.

DELETE Statement

This is used to delete rows from a table.

e.g.,

```
DELETE Employee WHERE Emp_Id = 7;
```

DELETE Employee
This command will delete all the rows from Employee table.

ORDER BY Clause

This clause is used to sort the result of a query in a specific order (ascending or descending).

By default sorting order is ascending.

```
SELECT Emp_Id, Emp_Name, City FROM Employee  
WHERE City = 'LUCKNOW'  
ORDER BY Emp_Id DESC;
```

GROUP BY Clause

It is used to divide the result set into groups. Grouping can be done by a column name or by the results of computed columns when using numeric data types.

- The HAVING clause can be used to set conditions for the GROUP BY clause.
- HAVING clause is similar to the WHERE clause, but having puts conditions on groups.
- WHERE clause places conditions on rows.
- WHERE clause can't include aggregate function, while HAVING conditions can do so.

e.g.,

```
SELECT Emp_Id, AVG (Salary)  
FROM Employee  
GROUP BY Emp_Id  
HAVING AVG (Salary) > 25000;
```

Aggregate Functions

Function	Description
SUM ()	It returns total sum of the values in a column.
AVG ()	It returns average of the values in a column.
COUNT ()	Provides number of non-null values in a column.
MIN () and MAX ()	Provides lowest and highest value respectively in a column.
COUNT (*)	Counts total number of rows in a table.

Joins

Joins are needed to retrieve data from two tables' related rows on the basis of some condition which satisfies both the tables. Mandatory condition to join is that atleast one set of column (s) should be taking values from same domain in each table.

Types of Joins

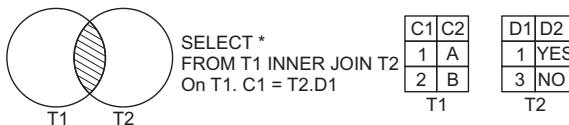
The Two types of joins are given below

Inner Join

Inner join is the most common join operation used in applications and can be regarded as the default join-type. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. *These may be further divided into three parts.*

- (i) Equi Join (satisfies equality condition)
 - (ii) Non-Equi Join (satisfies non-equality condition)
 - (iii) Self Join (one or more column assumes the same domain of values)
- Considers only pairs that satisfy the joining condition

Result is the intersection of the two tables.



Inner join in set T_1 and set T_2

Result set of T1 and T2

T1 · C1	C2	T2 · D1	D2
1	A	1	YES

Key Points

- ♦ The cross join does not apply any predicate to filter records from the joined table. Programmers can further filter the results of a cross join by using a WHERE clause.
- ♦ An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate
- ♦ A special case, a table (base table, view, or joined table) can JOIN to itself in a self-join.

Outer Join

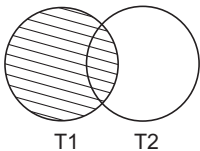
An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record-even if no other matching record exists.

Considers also the rows from table (s) even if they don't satisfy the joining condition

- (i) Right outer join
- (ii) Left outer join
- (iii) Full outer join

Left Outer Join

The result of a left outer join for table A and B always contains all records of the left table (A), even if the join condition does not find any matching record in the right table (B).



SELECT * FROM T1
Left Outer Join T2 ON T1.
ID = T2 . ID

ID	Name
1	Ram
2	Shyam
T1	

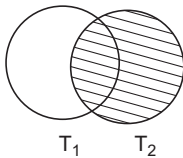
ID	Branch
1	IT
3	CS
Left Outer Join	

Result set of T1 and T2

T1.ID	Name	T2. ID	Branch
1	Ram	1	IT
2	Shyam	NULL	NULL

Right Outer Join

A right outer closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the right table will appear in the joined table at least once. If no matching with left table exists, NULL will appear.



SELECT * FROM T₁ RIGHT
OUTER JOIN T2 ON T1 . ID
=T2.ID

ID	Name	ID	Branch
1	Ram	1	IT
2	Shyam	3	CS

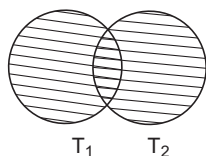
T1 T2
Right outer Join

Result set of T1 and T2

T1· ID	Name	T2 · ID	Branch
1	Ram	1	IT
NULL	NULL	3	CS

Full Outer Join

A full outer join combines the effect of applying both left and right outer joins. where records in the FULL OUTER JOIN table do not match, the result set will have NULL values for every column of the table that lacks a matching row. for those records that do match, as single row will be produced in the result set.



```
SELECT * FROM T1 FULL OUTER  
JOIN T2 ON T1 . ID=T2.ID
```

Result set of T1 and T2 (Using tables of previous example)

T1 · ID	Name	T2 · ID	Branch
1	Ram	1	IT
2	Shyam	NULL	NULL
NULL	NULL	3	CS

Cross Join (Cartesian Product)

Cross join returns the cartesian product of rows form tables in the join. It will produce rows which combine each row from the first table with each row from the second table.

```
Select * FROM T1, T2
```

Number of rows in result set = (Number of rows in table 1 × Number of rows in table 2)

Result set of T1 and T2 (Using previous tables T1 and T2)

ID	Name	ID	Branch
1	Ram	1	IT
2	Shyam	1	IT
1	Ram	3	CS
2	Shyam	3	CS

Key Points

- A programmer writes a JOIN predicate to identify the records for joining. If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table.
- A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).
- A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table.

Transaction Management

A sequence of many actions which are considered to be one atomic unit of work. A transaction is a collection of operations involving data items in a database. There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures.

Atomicity

Atomicity requires that each transaction is all or nothing. If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.

Consistency

If each transaction is consistent and the data base starts on as consistent, it ends up as consistent.

Isolation

Execution of one transaction is isolated from that of another transactions. It ensures that concurrent execution of transaction results in a system state that would be obtained if transaction were executed serially, *i.e.*, one after the other.

Durability

Durability means that once a transaction has been committed, it will remain even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently.

Key Points

- ♦ In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably.
 - ♦ In the context of databases, a single logical operation on the data is called a transaction.
 - ♦ A transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.
-

If a transaction commits, its effects persist.

- A transaction starts with any SQL statement and ends with a COMMIT or ROLLBACK.

- COMMIT statement makes changes permanent to the database.
- ROLLBACK statement reverses changes.
- A transaction includes one or more database access operations. These can include insertion, deletion, modification or retrieval operations.

Database Access Operations

The basic database access operations are

- **Read-item (X)** Reads a database item named X into a program variable or $R(X)$.
- **Write-item (X)** Writes the value of program variable X into the database item named X or $W(X)$.

Classification of a Database System According to the Number of Users

Single User

A DBMS is single user, if at most one user at a time can use the system.

Multiusers

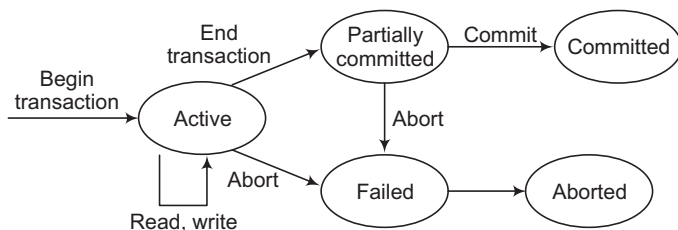
A DBMS is multiusers, if many user can use the system and hence access the database concurrently.

e.g., An airline reservation system is used by hundreds of travel agents and reservation clerks submit transactions concurrently to the system.

Transaction States

The following are the different states in transaction processing in a database system

1. Active
2. Partially committed
3. Failed
4. Aborted



State transition diagrams

Active This is the initial state. The transaction stay in this state while it is executing.

Partially Committed This is the state after statement of the transaction is executed.

Failed After the discovery that normal execution can no longer proceed.

Aborted The state after the transaction has been rolled back and the database has been resorted to its state prior to the start of the transaction.

Isolation Levels

If every transaction does not make its updates visible to other transaction until it is committed, so isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks.

There have been attempts to define the level of isolation of a transaction.

Level-0 Isolation A transaction is said to have level-0 isolation, if it does not overwrite the dirty reads of higher-level transactions.

Level-1 Isolation Level-1 isolation has no lost updates.

Level-2 Isolation Level-2 isolation has no lost updates and no dirty reads.

Level-3 Isolation Level-3 isolation (also true isolation) has repeatable reads.

Concurrency Control

Process of managing simultaneous execution of transactions in a shared database, is known as **concurrency control**. Basically, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

Need of Concurrency Control

Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.

Lost Update

This problem occurs when two transactions that access the same database items, have their operations interleaved in a way that makes the value of some database items incorrect.

Dirty Read Problems

This problem occurs when one transaction reads changes the value while the other reads the value before committing or rolling back by the first transaction.

Inconsistent Retrievals

This problem occurs when a transaction accesses data before and after another transaction(s) finish working with such data.

We need concurrence control, when

- The amount of data is sufficiently great that at any time only fraction of the data can be in primary memory and rest should be swapped from secondary memory as needed.
- Even if the entire database can be present in primary memory, there may be multiple processes.

Key Points

- A failure in concurrency control can result in data corruption from torn read or write operations.
- DBMS need to deal also with concurrency control issues not typical just to database transactions but rather to operating systems in general.
- Concurrency control is an essential element for correctness in any system where two database transactions or more, executed with time overlap, can access the same data, e.g., virtually in any general purpose database system.

Schedule

A schedule (or history) is a model to describe execution of transactions running in the system. When multiple transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from the various transactions is known as a schedule or we can say that a schedule is a sequence of read, write, abort and commit operations from a set of transactions.

The following is an example of a schedule

T ₁	T ₂	T ₃
R(X) W(X) Commit	R(Y) W(Y) Commit	R(Z) W(Z) Commit

The above schedule consists of three transactions T_1, T_2 and T_3 . The first transaction T_1 reads and writes to object X and then commits. Then, T_2 reads and writes to object Y and commits and finally T_3 reads and writes to object Z and commits.

Classification of Schedules Based on Serializability

The schedules can be classified as

Serial Schedule

A schedule in which the different transactions are not interleaved (i.e., transactions are executed from start to finish one-by-one).

Serial Schedule		Serial Schedule	
T_1	T_2	T_1	T_2
	$R(A)$ $W(A)$	$R(B)$ $W(B)$	
$R(B)$ $W(B)$			$R(A)$ $W(A)$

Complete Schedule

A schedule that contains either a commit or an abort action for each transaction.

Note Consequently, a complete schedule will not contain any active transaction at the end of the schedule.

Complete Schedule		Complete Schedule		Complete Schedule	
T_1	T_2	T_1	T_2	T_1	T_2
$R(A)$		$R(A)$		$R(A)$	
	$R(B)$	$W(A)$		$W(A)$	
$W(A)$			$R(B)$	Commit	
	$W(B)$	Commit			$R(B)$
Commit			$W(B)$		$W(B)$
	Abort		Abort		Abort

Non-serial Schedules

Non-serial schedules are interleaved schedules and these schedules improve performance of system (i.e., throughput and response time). But concurrency or interleaving operations in schedules, might lead the database to an inconsistent state.

We need to seek to identify schedules that are

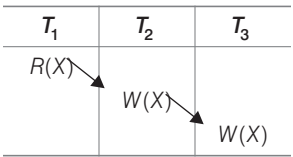
- (i) As fast as interleaved schedules.
- (ii) As consistent as serial schedules.

Conflicting Operations When two or more transactions in a non-serial schedule execute concurrently, then there may be some conflicting operations.

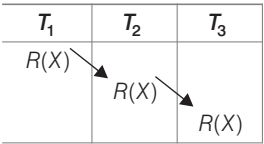
Two operations are said to be conflicting, if they satisfy all of the following conditions

- The operations belong to different transactions.
- Atleast one of the operations is a write operation.
- The operations access the same object or item.

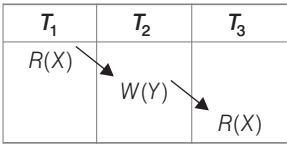
The following set of operations is conflicting



While the following sets of operations are not conflicting



//No write on same object



//No write on same object

Key Points

- ♦ If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, hence the schedules are not conflict equivalent.
- ♦ In view equivalence respective transactions in the two schedules read and write the same data values while in conflict equivalence, respective transactions in two schedules have the same order of conflicting operations.

Serializable Schedule

A schedule S of n transactions is serializable, if it is equivalent to some serial schedule of the same n transactions.

A non-serial schedule S is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule.

There are two types of serializable schedule

- (i) Conflict serializable schedule
- (ii) View serializable schedule

Conflict Serializable Schedule

When the schedule (S) is conflict equivalent to some serial schedule (S') , then that schedule is called as conflict serializable schedule. In such a case, we can reorder the non-conflicting operations in S until we form the equivalent serial schedule S' .

Serial schedule		Serializable schedule A		Serializable schedule B	
T_1	T_2	T_1	T_2	T_1	T_2
$R(A)$ $W(A)$ $R(B)$ $W(B)$		$R(A)$ $W(A)$	$R(A)$ $W(A)$	$R(A)$	$R(A)$ $W(A)$
	$R(A)$ $W(A)$	$R(B)$ $W(B)$	$R(B)$ $W(B)$	$R(B)$ $W(B)$	$R(B)$ $W(B)$
	$R(B)$ $W(B)$			$W(A)$ $R(B)$ $W(B)$	

Conflict Equivalence

- The schedules S_1 and S_2 are said to be conflict equivalent, if the following conditions are satisfied
- Both schedules S_1 and S_2 involve the same set of transactions (including ordering of operations within each transaction).
 - The order of each pair of conflicting actions in S_1 and S_2 are the same.

Testing for Conflict Serializability of a Schedule

There is a simple algorithm that can be used to test a schedule for conflict serializability. This algorithm constructs a precedence graph (or serialization graph), which is a directed graph.

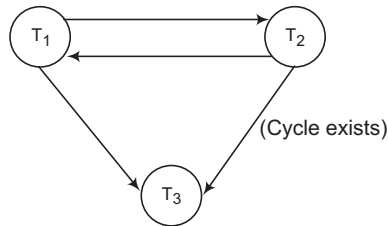
A precedence graph for a schedule S contains

- (i) A node for each committed transaction in S .
- (ii) An edge from T_i to T_j , if an action of T_i precedes and conflicts with one of T_j 's operations.

T_1	T_2	T_3
$R(A)$		
	$W(A)$	
$W(A)$		
		$W(A)$

Conflict serializability

A schedule S is conflict serializable if and only if its precedence graphs is acyclic.



Precedence graph

Note The above example of schedule is not conflict serializable schedule. However, in general, several serial schedules can be equivalent to any serializable schedule S , if the precedence graph for S has no cycle and if the precedence graph has a cycle, it is easy to show that we cannot create any equivalent serial schedule, so S is not serializable.

Key Points

- ✦ Serial schedule is slower but guarantees consistency (correctness).
- ✦ Every serial schedule is a serializable schedule but not every serializable schedule is serial schedule.
- ✦ Being serializable implies that
The schedule is a correct schedule, if

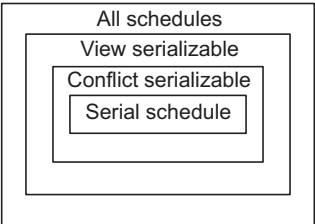
It will leave the database in a consistent state and the interleaving is appropriate and will result in a state as if the transaction were serially executed.

A schedule is a efficient (interleaved) schedule.

View Serializable Schedule

A schedule is view serializable, if it is view equivalent to some serial schedule.

Conflict serializable \Rightarrow View serializable, but not vice-versa.



View Equivalence

Two schedules S_1 and S_2 are view equivalent,

- (i) T_i reads initial value of database object A in schedule S_1 , then T_i also reads initial value of database object A in schedule S_2 .
- (ii) T_i reads value of A written by T_j in schedule S_1 , then T_i also reads value of A written by T_j in schedule S_2 .
- (iii) T_i writes final value of A in schedule S_1 , then T_i also writes final value of A in S_2 .

Schedule S_1			Schedule S_2		
T_1	T_2	T_3	T_1	T_2	T_3
$R(A)$			$R(A)$		
	$W(A)$			$W(A)$	
$W(A)$		$W(A)$			$W(A)$

In the above example, both schedules S_1 and S_2 are view equivalent. So, they are view serializable schedules. But S_1 schedule is not conflict serializable schedule and S_2 is not conflict serializable schedule because cycle is not formed.

Classification of Schedules Based on Recoverability

- In recoverability, there is need to address the effect of transaction failures on concurrently running transactions.
- Once a transaction T is committed, it should never be necessary to rollback T .
- The schedules those meet this criterion are called recoverable schedules and those do not, are called **non-recoverable**.

Recoverable Schedule

Once a transaction T is committed, it should never be necessary to rollback T . The schedules under this criteria are called recoverable schedules and those do not, are called non-recoverable.

A schedule S is recoverable, if no transaction T in S commits until all transactions T' , that have written an item that T reads, have committed.

Initial Recoverable Schedule	
T_1	T_2
$R(X)$ $W(X)$	$R(X)$ $W(X)$ Commit
Abort	

(This schedule is not recoverable because T_2 made a dirty read and committed before T_1 , T_1 should have committed first.)

The above given schedule is non-recoverable because when the recovery manager rolls back T_1 transaction, then X gets its initial value before updation. But T_2 has already utilised the wrong value of X that was updated by T_1 and T_2 committed. Now, database is consequently in an inconsistent state.

Composition of Recoverable

Initial Non-recoverable Schedule	
T_1	T_2
$R(X)$ $W(X)$	$R(X)$ $W(X)$ Commit
Abort	

Recoverable Schedule (A)		Recoverable Schedule (B)	
T_1	T_1	T_1	T_2
$R(X)$ $W(X)$	$R(X)$ $W(X)$ Commit	$R(X)$ $W(X)$ Abort	$R(X)$ $W(X)$ Commit
Abort		Remove dirty read (Recoverable) not serializable schedule (Order of conflicting actions has changed)	

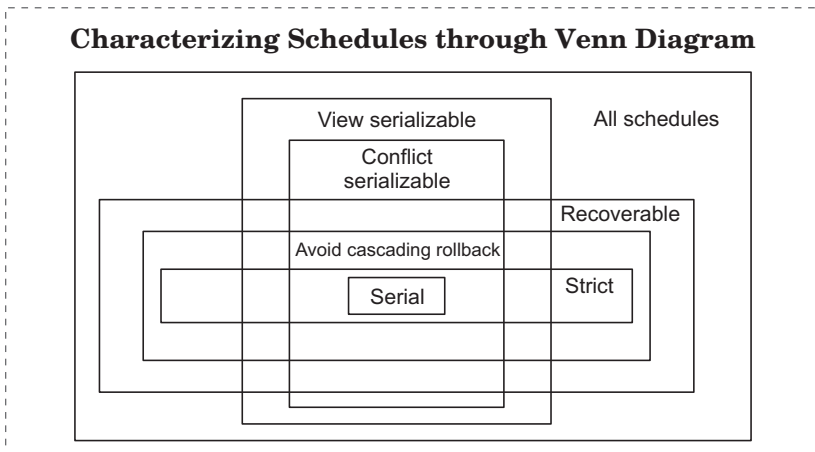
Cascadeless Schedule

These schedules avoid cascading rollbacks. Even, if a schedule is recoverable to recover correctly from failure of transaction T_i , we may have to rollback several transactions. This phenomenon in which a single transaction failure which leads to a series of transaction rollbacks, is called **cascading rollbacks**.
Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those, where cascading rollback cannot occur. Such schedules are called **cascadeless schedules**.

Cascadeless Schedule	
T_1	T_2
$R(X)$ $W(X)$ $R(Y)$ $W(Y)$ (a) Rollback Abort	(b) Now T_2 reads the value of X that existed before T_1 started. $R(X)$ $W(X)$ Commit

Strict Schedule

- A schedule is strict,
- If overriding of uncommitted data is not allowed.
 - Formally, if it satisfies the following conditions
 - (i) T_j reads a data item X after T_i has terminated (aborted or committed).
 - (ii) T_j writes a data item X after T_i has terminated (aborted or committed).



Concurrency Control with Locking

- Previously, we have characterised transaction schedules based on serializability and recoverability.
- If concurrency control with locking technique is used, then locks prevent multiple transactions from accessing the items concurrently.
- Access on data only, if TA 'has lock' on data.
- Transactions request and release locks.
- Schedule allows or differs operations based on lock table.

Lock

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. (e.g., read lock, write lock). Locks are used as means of synchronizing the access by concurrent transactions to the database items. There is one lock per database item.

Problems Arising with Locks

There are two problems which are arised when using locks to control the concurrency among transactions

Deadlock Two or more competing transactions are waiting for each other to complete to obtain a missing lock.

Starvation A transaction is continually denying the access to a given data item.

Purpose of Concurrency Control with Locking

- To enforce isolation among conflicting transactions.
- To preserve database consistency.
- To resolve read-write, write-read and write-write conflicts.

Locking is an operation that secures

- Permission to read
- Permission to write a data item

Lock (X) Data item X is locked on behalf of the requesting transaction. Unlocking is an operation which removes these permissions from the data item.

Unlock (X) Data item X is made available to all other transactions.

Types of Locks

The two types of Locks are given below

Binary Locks

A binary lock has two states

Locked/Unlocked

If a database item X is locked, then X cannot be accessed by any other database operation.

Shared/ Exclusive Locks (Read/Write Locks)

There are three states

Read locked/Write locked/Unlocked

Several transactions can access the same item X for reading (shared lock), however if any of the transactions want to write the item X , the transaction must acquire an exclusive lock on the item.

Note Using binary locks or read/write locks in transactions, does not guarantee serializability of schedules on its own.

Key Points

- ♦ Lock and unlock are atomic operations (all or nothing).
- ♦ If every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable.

Guaranteeing Serializability by Two-phase Locking

A transaction is said to follow the **two-phase locking protocol**, if all locking operations precede the first unlock operation in the transaction.

Such a transaction can be divided into two phases

Expanding or Growing Phase During which new locks on items can be acquired but none can be released.

Shrinking Phase During which existing locks can be released but no new locks can be acquired.

Variations of Two-phase Locking (2PL) Technique

Basic 2PL (2 Phase Locking) Transaction locks data items incrementally. This may cause the problem of deadlock.

Conservative 2PL (Static 2PL) Prevents deadlock by locking all desired data items before transaction begins execution. However, it is difficult to use in practice because of the need to predeclare the read-set and write-set, which is not possible in most situations.

Strict 2PL A more stricter version of basic algorithm, where unlocking of write lock is performed after a transaction terminates (commits or aborts and rolled back). Hence, no other transaction can read or write an item that is written by transaction.

Rigorous 2PL Strict 2PL is not deadlock free. A more restrictive variation of strict 2PL, is rigorous 2PL, which also guarantees strict schedule. In this variation, a transaction does not release any of its locks (exclusive or shared) until after it commits or aborts.

Deadlock

A deadlock is a condition in which two or more transaction are waiting for each other deadlock (T_1 and T_2). *An example is given below*

T_1	T_2
Read-lock (Y) $R(Y)$	Read-lock (X) $R(X)$
Write-Lock (X) (Waits for X)	Write-Lock (Y) (Waits for Y)

Deadlock Prevention

A transaction locks all data items, it refers to before it begins execution. This way of locking prevents deadlock, since a transaction never waits for a data item. The conservative two-phase locking uses this approach.

Deadlock Detection and Resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled back.

Key Points

- ♦ A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph.
 - ♦ When a chain like T_i waits for T_j , T_j waits for T_k and T_k waits for T_i or T_j occurs, then this creates a cycle. One of the transaction of the cycle is selected and rolled back.
-

Deadlock Avoidance

There are many variations of two-phase locking algorithm. Some avoid deadlock by not letting the cycle to complete. This is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.

Following schemes use transaction times-tamps for the sake of deadlock avoidance

Wait-die scheme (Non-preemptive)

Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead. A transaction may die several times before acquiring the needed data item.

Wound-wait scheme (Preemptive)

Older transaction bounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones. May be fewer rollbacks than wait-die scheme.

Starvation

Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further. In a deadlock resolution scheme, it is possible that the same transaction may consistently be selected as victim and rolled back.

Time-stamp Based Concurrency Control Algorithm

This is a different approach that guarantees serializability involves using transaction time-stamps to order transaction execution for an equivalent serial schedule.

Time-stamp

A time-stamp is a unique identifier created by the DBMS to identify a transaction.

- Key Points
- Time-stamp is a monotonically increasing variable (integer) indicating the age of an operation or a transaction.
 - A larger time-stamp value indicates a more recent event or operation.

Starvation versus Deadlock

Starvation	Deadlock
Starvation happens if same transaction is always chosen as victim.	A deadlock is a condition in which two or more transaction are waiting for each other.
It occurs if the waiting scheme for locked items is unfair, giving priority to some transactions over other.	A situation where two or more transactions are unable to proceed because each is waiting for one of the other to do something.
Starvation is also known as lived lock.	Deadlock is also known as circular waiting.
Avoidance Switch priorities so that every thread has a chance to have high priority.	Avoidance Acquire locks are predefined order.
Use FIFO order among competing request.	Acquire locks at one before starting.
It means that transaction goes in a state where transaction never progress.	It is a situation where transaction are waiting for each other.

The algorithm associates with each database item X with two Time Stamp (TS) values

Read_TS (X)

The read time stamp of item X; this is the largest time-stamp among all the time-stamps of transactions that have successfully read item X.

Write_TS (X)

The write time-stamp of item X ; this is the largest time-stamp among all the time-stamps of transactions that have successfully written item X .

There are two Time-Stamp based Concurrency Control Algorithm

Basic Time-stamp Ordering (TO)

Whenever some transaction T tries to issue a $R(X)$ or a $W(X)$ operation, the basic time-stamp ordering algorithm compares the time-stamp of T with $read_TS(X)$ and $write_TS(X)$ to ensure that the time-stamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted. If T is aborted and rolled back, any transaction T_1 that may have used a value written by T must also be rolled back. Similarly, any transaction T_2 that may have used a value written by T_1 must also be rolled back and so on. This effect is known as cascading rollback.

The concurrency control algorithm must check whether conflicting operations violate the time-stamp ordering in the following two cases

Transaction T issues a $W(X)$ operation

If $read_TS(X) > TS(T)$ or if $write_TS(X) > TS(T)$, then an younger transaction has already read the data item, so abort and roll back T and reject the operation.

If the condition in above part does not exist, then execute $W(X)$ of T and set $write_TS(X)$ to $TS(T)$.

Transaction T issues a $R(X)$ operation

If $write_TS(X) > TS(T)$, then an younger transaction has already written to the data item, so abort and rollback T and reject the operation.

If $write_TS(X) \leq TS(T)$, then execute $R(X)$ of T and set $read_TS(T)$ to the larger of $TS(T)$ and the current $read_TS(X)$.

Strict Time-stamp Ordering (TO)

A variation of basic time-stamp ordering called strict time-stamp ordering, ensures that the schedules are both strict (for easy recoverability) and serializable (conflict).

Transaction T issues a $W(X)$ operation If $TS(T) > read_TS(X)$, then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

Transaction T issues a $R(X)$ operation If $TS(T) > write_TS(X)$, then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

Thomas's Write Rule

- If $\text{read_TS}(X) > \text{TS}(T)$, then abort and rollback T and reject the operation.
 - If $\text{write_TS}(X) > \text{TS}(T)$, then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
- If the conditions given in (i) and (ii) above do not occur, then execute $W(X)$ of T and set $\text{write_TS}(X)$ to $\text{TS}(T)$.

Multiversion Concurrency Control Techniques

This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus, unlike other mechanisms a read operation in this mechanism is never rejected.

Side effect Significantly more storage is required to maintain multiple versions.

Validation Based Concurrency Control Protocol

In this, execution of transaction T_i is done in three phases.

Read and Execution Phase Transaction T_i writes only to temporary local variables.

Validation Phase Transaction T_i performs a validation test to determine, if local variables can be written without violating serializability.

Write Phase If T_i is validated, the updates are applied to the database, otherwise T_i is rolled back.

Note The above three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.

Each transaction T_i has three time-stamps

Start (T_i) The time when T_i started its execution.

Validation (T_i) The time when T_i entered its validation phase.

Finish (T_i) The time when T_i finished its write phase.

Serializability order is determined by time-stamp given at validation time to increase concurrency. Thus, $\text{TS}(T_i)$ is given the value of validation (T_i).

For all T_j with $\text{TS}(T_i) < \text{TS}(T_j)$ either one of the following conditions holds

- (i) $\text{Finish}(T_i) < \text{Start}(T_j)$ (ii) $\text{Start}(T_j) < \text{Finish}(T_i) < \text{Validation}(T_j)$

and the set of data items written by T_i does not intersect with the set of data items read by T_j . Then, validation succeeds and T_j can be committed. Otherwise, validation fails and T_j is aborted.

Multiple Granularity

Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones. It can be represented graphically as a tree. When a transaction locks a node in the tree explicitly, it implicitly locks all the nodes descendents in the same node.

Coarse Granularity The larger the data item size, the lower the degree of concurrency.

Fine Granularity The smaller the data item size, the more locks to be managed and stored and the more lock/unlock operations needed.

Key Points

- ♦ Multiple users can access databases and use computer system simultaneously because of the concept of multiprogramming, which allows the computer to execute multiple programs or processes at the same time.
 - ♦ If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible.
 - ♦ While one transaction is waiting for a page to be read from disk, the CPU can process another transaction. This may increase system throughput (the average number of transactions completed in a given time).
 - ♦ Interleaved execution of a short transaction with a long transaction allows the short transaction to complete first. This gives improved response time (average time taken to complete a transaction).
 - ♦ A transaction may be incomplete because the (database) system crashes or because it is aborted by either the system or the user (or application).
 - ♦ Complete transactions are committed.
 - ♦ Consistency and isolation are primarily the responsibility of a scheduler. Atomicity and durability are primarily responsibility of recovery manager.
-

Interleaving

Multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process and so on. A process is resumed at the point, where it was suspended, whenever it gets its turn to use the CPU again. Hence, this process is actually known as **interleaving**.

File Structures

File structure or organisation refers to the relationship of the key of the record to the physical location of that record in the computer file.

Disk Storage

Databases must be stored physically as files of records, which are typically stored on some computer storage medium. The DBMS software can then retrieve, update and process this data as needed.

Several aspects of storage media must be taken into account

- (i) Speed with which data can be accessed.
- (ii) Cost per unit of data
- (iii) Reliability
- Data loss on power failure or system crash.
- Physical failure of the storage device.
- So, we can differentiate storage into

Volatile Storage

Losses contents when power is switched off.

Non-volatile Storage

Contents persist even when power is switched off.

Category of Computer Storage Media

Computer storage media form a storage hierarchy that includes two main categories.

- **Primary Storage** This category includes storage media that can be operated on directly by the computer Central Processing Unit (CPU), such as the computer main memory and smaller but faster cache memories. Primary storage usually provides fast access to data but is of limited storage capacity.
- **Secondary Storage** This category includes magnetic disks, optical disks and tapes. These devices usually have a larger capacity, cost less and provides slower access to data than do primary storage devices. Data in secondary storage cannot be processed directly by the CPU.

Characteristics of Secondary Storage Devices

- (i) Random access *versus* sequential access
- (ii) Read-write, write-once, read-only
- (iii) Character *versus* block data access

Storage of Databases

Need data to be stored permanently or persistently over long periods to time. Cost of storage per unit of data is an order of magnitude, less for disk secondary storage than for primary storage.

Magnetic Hard Disk Mechanism

Magnetic disks are used for storing large amount of data. The most basic unit of data on the disk is a single bit of information. All disks are made of magnetic material shaped as a thin circular disk and protected by a plastic or acylic cover. A disk is single sided, if it stores information on one of its surfaces only and double-sided, if both surfaces are used.

Key Points

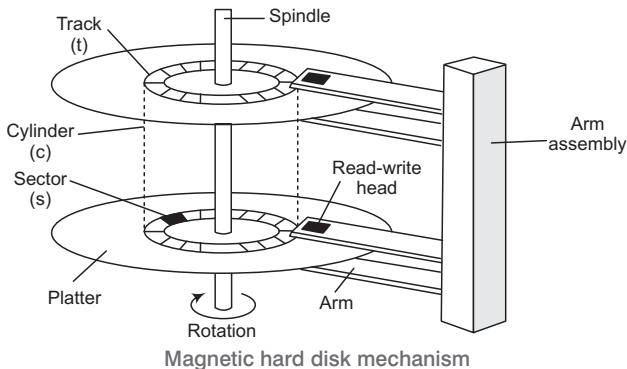
- ♦ To increase storage capacity, disks are assembled into a disk pack, which may include many disks.
- ♦ Information is stored on disk surface in concentric circles, each circle is called a track.
- ♦ The disk surfaces are called platter surface.

Read-write Head

- Positioned very close to the platter surface (touching it).
- Reads or writes magnetically encoded information.

Each Track is Divided into Sectors

- A sector is the smallest unit of data that can be read or written.
- Sector size typically 512 bytes.
- Typical sectors per track : 500 (on inner tracks) to 1000 (on outer tracks).



Performance Measures on Disks

Access Time The time it takes from when a read or write request is issued to when data transfer begins. *It consists of*

- **Seek time** Time it takes to reposition the arm over the correct track. Average seek time is 1/2 of the worst case seek time.
- **Rotational latency** Time it takes for the sector to be accessed to appear under the head. Average latency is 1/2 of the worst case latency.

Data Transfer Rate The rate at which the data can be retrieved from or stored to the disk.

Mean Time To Failure (MTTF) The average time, the disk is expected to run continuously without any failure.

Optimization of Disk-Block Access

Block A contiguous sequence of sectors from a single track. Data is transferred between disk and main memory in blocks. Sizes range from 512 bytes to several kilobytes. *Blocks are of two types*

- **Smaller blocks** more transfers from disk.
- **Larger blocks** more space wasted due to partially filled blocks.

Disk-arm Scheduling This algorithm orders pending accesses to tracks so that disk arm movement is minimized.

Elevator Algorithm In this algorithm, move disk arm in one direction (from outer to inner tracks or *vice-versa*), processing next request in that direction, till no more request in that direction, then reverse direction and repeat.

RAID (Redundant Arrays of Independent Disks)

The choice of disk structure is very important in databases. *Important factors, besides price are*

- | | | |
|--------------|------------|-------------------|
| (i) Capacity | (ii) Speed | (iii) Reliability |
|--------------|------------|-------------------|

It is a disk organisations technique that manage a large numbers of disks, *providing a view of a single disk of*

- High capacity and high speed by using multiple disks in parallel and
- High reliability by storing data redundantly so that data can be recovered even if a disk fails.

Storage Access in Databases

A database file is partitioned into fixed length storage units called blocks. Blocks are units of both storage allocation and data transfer. Database system seeks to minimize the number of block transfers between the disk and memory. The number of disk accesses can be reduced by keeping as many blocks as possible in main memory.

Buffer

It is a portion of main memory that is available to store copies of disk blocks when several blocks need to be transferred from disk to main memory and all the block addresses are known, several buffers can be reserved in main memory to speed up the transfer while one buffer is being read or written, the CPU can process data in the other buffer.

Buffer Manager

It is a subsystem which is responsible for allocating buffer space in main memory.

Programs call on the buffer manager when they need a block from disk.

- (i) If the block is already in the buffer, buffer manager returns the address of the block in main memory.
- (ii) If the block is not in the buffer, the buffer manager

Space in the Buffer

Allocates space in the buffer for the block.

- Replacing (throwing out) some other block, if required to make space for the new block.
- Replaced the block written back to disk only if it was modified, since the most recent time that it was written to/fetched from the disk.

Reads the block from the disk to the buffer and returns the address of the block in main memory to requester.

Buffer Replacement Policies Most operating systems replace the block that is Least Recently Used (LRU strategy).

Most Recently Used (MRU) Strategy System must pin the blocks currently being processed. After the final tuple of that block has been processed, the block is uninned and it becomes the most recently used blocks.

Pinned Block Memory block that is not allowed to be written back to disk.

File Organisation

The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields. Data is usually stored in the form of records. Records usually describe entities and their attributes. e.g., an employee record represents an employee entity and each field value in the record specifies some attributes of that employee, such as Name, Birth-date, Salary or Supervisor.

If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed length records. If different records in the file have different sizes, the file is said to be made up of variable length records.

Spanned versus Unspanned Records

- The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory. When the block size is larger than the record size, each block will contain numerous records, although some of the files may have unusually large records that cannot fit in one block.
- Suppose that block size is B bytes. For a file of fixed length records of size R bytes, with $B \geq R$, then we can fit $bfr = \left\lfloor \frac{B}{R} \right\rfloor$ records per block. The value of bfr is called the blocking factor.
- In general, R may not divide B exactly, so we have some unused space in each block equal to $B - (bfr * R)$ bytes.
- To utilize this unused space, we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record. This organisation is called spanned.
- If records are not allowed to cross block boundaries, the organisation is called unspanned.

Allocating File Blocks on Disk

There are several standard techniques for allocating the blocks of a file on disk

Contiguous Allocation The file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast.

Linked Allocation In this, each file contains a pointer to the next file block.

Indexed Allocation Where one or more index blocks contain pointers to the actual file blocks.

File Headers

A file header or file descriptor contains information about a file that is needed by the system programs that access the file records.

Operations on Files

Operations on files are given below

Retrieval Operations These do not change any data in the files but only locate certain records so that their field values can be examined and processed.

Update Operations These change the file by insertion or deletion of records or by modification of field values.

Open Prepares the file for reading or writing. Set the file pointer to the beginning of the file.

Reset Sets the file pointer of an open file to the beginning of the file.

Find (or **Locate**) Searches for the first record that satisfies a search condition. Transfers the block containing that record into a main memory buffer (if it is not already there).

Read (or **less Get**) Copies the current record from the buffer to a program variable in the user program.

FindNext Searches for the next record in the file that satisfies the search condition.

Delete Deletes the current record and updates the file on disk to reflect the deletion.

Modify Modifies some field values for the current record and updates the file on disk to reflect the modification.

Insert Insert a new record in the file by locating the block, where the record is to be inserted, transferring that block into the main memory buffer, writing the record into the buffer and writing the buffer to disk to reflect the insertion.

Close Completes the file access by releasing the buffers and performing any other needed cleanup operations.

Files of Unordered Records (Heap Files)

In the simplest type of organization records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organisation is called a heap or pile file.

This organisation is often used with additional access paths, such as the secondary indexes.

In this type of organisation, inserting a new record is very efficient. Linear search is used to search a record.

Files of Ordered Records (Sorted Files)

- We can physically order the records of a file on disk based on the values of one of their fields called the ordering field. This leads to an ordered or sequential file.
- If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record, then the field is called the ordering key for the file. Binary searching is used to search a record.

Hashing Techniques

Another type of primary file organisation is based on hashing which provides very fast access to records under certain search conditions. This organisation is usually called a hash file. A hash file has also been called a direct file.

Key Points

- ♦ The search condition must be an equality condition on a single field, called the hash field.
- Hash field = Key field

 \Rightarrow

Hash key
- ♦ Hash function is used to determine page address for storing records. This is chosen to provide most even distribution of records and tends to minimum collisions.

There are various techniques for hashing

Internal Hashing

For internal files, hashing is typically implemented as a hash table through the use of an array of records and the array index range is from 0 to $M - 1$, then we have M slots in an array.

One common hash function is

$$h(K) = K \bmod M$$

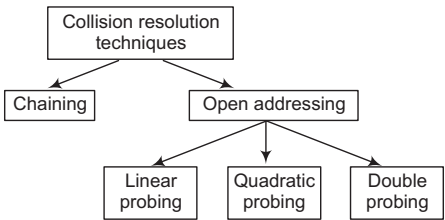
Which returns the remainder of an integer hash field value K after division by M .

Collision

Hash function does not calculate unique address for two or more records. Here, a collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In

this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called **collision resolutions**.

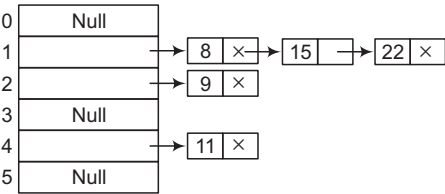
There are number of methods for collision resolution



Classification of collision resolution

Chaining

In this method, all the elements, where keys hash to the same hash table slot are put in linked list manner.



Collision Resolution by Open Addressing

In open addressing, the keys to be hashed is to put in the separate location of the hash table. Each location contains some key or the some other character to indicate that the particular location is free.

In this method to insert key into the table, we simply hash the key using the hash function. If the space is available, then insert the key into the hash table location, otherwise search the location in the forward direction of the table to find the slot in a systematic manner. The process of finding the slot in the hash table is called **probing**.

Linear probing The linear probing use the following hash function
$$h(k,i) = [h'(k) + i] \bmod m \text{ for } i = 0, 1, 2, \dots m - 1,$$
where m is the size of the hash table, $h'(k) = k \bmod m$, the basic function, and i = The probe number.

Quadratic probing The quadratic probing uses the following hash function
$$h(k,i) = [h'(k) + c_1i + c_2i^2] \bmod m \text{ for } i = 0, 1, \dots m - 1$$

where, m = size of hash table

$h'(k) = k \bmod m$, the basic hash function

c_1 and $c_2 \neq 0$ are auxiliary and i = the probe number.

Double hashing In double hashing, second hashing function H' is used for resolving a collision. Suppose a record R with key K has the hash address

$$H(K) = h \text{ and } H'(K) = h' \neq m$$

Then, we linearly search the location with addresses

$$h, h + h', h + 2h', h + 3h' \dots$$

If m is prime number, then the above sequence will access all the locations in the table T .

External Hashing for Disk Files

Hashing for disk files is called external hashing. To suit the characteristics of disk storage, the target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hashing function maps a key into a relative bucket number, rather than assigning an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address. The collisions problem is less severe with buckets.

Indexing Structures for Files

Indexing mechanism are used to optimize certain accesses to data (records) managed in files. e.g., the author catalog in a library is a type of index. Search key (definition) attribute or combination of attributes used to look-up records in a file.

An index file consists of records (called index entries) of the form.

Search key value	Pointer of block in data file
------------------	-------------------------------

Indexing structures for files

Index files are typically much smaller than the original file because only the values for search key and pointer are stored. The most prevalent types of indexes are based on ordered files (single-level indexes) and tree data structures (multilevel indexes).

Types of Single Level Ordered Indexes

In an ordered index file, index entries are stored sorted by the search key value. *There are several types of ordered Indexes*

Primary Index

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field called the primary key of the data file and the second field is a pointer to a disk block (a block address).

Key Points

- ♦ There is one index entry in the index file for each block in the data file.
 - ♦ Indexes can also be characterised as dense or sparse.
 - ♦ **Dense index** A dense index has an index entry for every search key value in the data file.
 - ♦ **Sparse index** A sparse index (non-dense), on the other hand has index entries for only some of the search values.
 - ♦ A primary index is a non-dense (sparse) index, since it includes an entry for each disk block of the data file rather than for every search value.
-

Clustering Index

If file records are physically ordered on a non-key field which does not have a distinct value for each record that field is called the clustering field. We can create a different type of index, called a clustering index, to speed up retrieval of records that have the same value for the clustering field.

Key Points

- ♦ A clustering index is also an ordered file with two fields. The first field is of the same type as the clustering field of the data file.
 - ♦ The record field in the clustering index is a block pointer.
 - ♦ A clustering index is another example of a non-dense index.
-

Secondary Index

A secondary index provides a secondary means of accessing a file for which some primary access already exists. The secondary index may be on a field which is a candidate key and has a unique value in every record or a non-key with duplicate values. The index is an ordered file with two fields. The first field is of the same data type as some non-ordering field of the data file that is an indexing field. The second field is either a block pointer or a record pointer.

A secondary index usually needs more storage space and longer search time than does a primary index.

Multilevel Indexes

The idea behind a multilevel index is to reduce the part of the index. A multilevel index considers the index file, which will be referred now as the first (or base) level of a multilevel index. Therefore, we can create a primary index for the first level; this index to the first level is called the second level of the multilevel index and so on.

Dynamic Multilevel Indexes Using B-Trees and B⁺-Trees

There are two multilevel indexes

B-Trees

- When data volume is large and does not fit in memory, an extension of the binary search tree to disk based environment is the B-tree.
- In fact, since the B-tree is always balanced (all leaf nodes appear at the same level), it is an extension of the balanced binary search tree.
- The problem which the B-tree aims to solve is given a large collection of objects, each having a key and a value, design a disk based index structure which efficiently supports query and update.
- A B-tree of order p , when used as an access structure on a key field to search for records in a data file, can be defined as follows

- (i) Each internal node in the B-tree is of the form

$$\langle P_1, \langle K_1, P_{f_1} \rangle, P_2, \langle K_2, P_{f_2} \rangle, \dots, \langle K_{q-1}, P_{f_{q-1}} \rangle, P_q \rangle$$

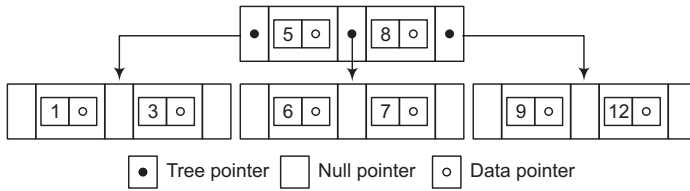
where, $q \leq p$.

Each P_i is a tree pointer to another node in the B-tree.

Each P_{f_j} is a data pointer to the record whose search key field value is equal to K_j .

- (ii) Within each node, $K_1 < K_2 < \dots < K_{q-1}$.
- (iii) Each node has at most p tree pointers.
- (iv) Each node, except the root and leaf nodes, has atleast $\lceil (p/2) \rceil$ tree pointers.
- (v) A node within q tree pointers $q \leq p$, has $q - 1$ search key field values (and hence has $q - 1$ data pointers).

e.g., A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



B⁺ Trees

- It is the variation of the B-tree data structure.
- In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer. In a B⁺-tree, data pointers are stored only at the leaf nodes of the tree. Hence, the structure of the leaf nodes differs from the structure of internal nodes.
- The pointers in the internal nodes are tree pointers to blocks that are tree nodes whereas the pointers in leaf nodes are data pointers.

B⁺ Tree's Structure

The structure of the B⁺-tree of order p is as follows

- Each internal node is of the form $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$
Where, $q \leq p$ and each P_i is a tree pointer.
- Within each internal node, $K_1 < K_2 < K_3 \dots < K_{q-1}$.
- Each internal node has at most p tree pointers and except root, has atleast $\lceil (p/2) \rceil$ tree pointers. The root node has atleast two tree pointers, if it is an internal node.
- Each leaf node is of the form.
$$\langle \langle K_1, P_{f_1} \rangle, \langle K_2, P_{f_2} \rangle, \dots, \langle K_{q-1}, P_{f_{q-1}} \rangle, P_{\text{next}} \rangle$$

Where, $q \leq p$, each P_{f_i} is a data pointer and P_{next} points to the next leaf node of the B⁺-trees.