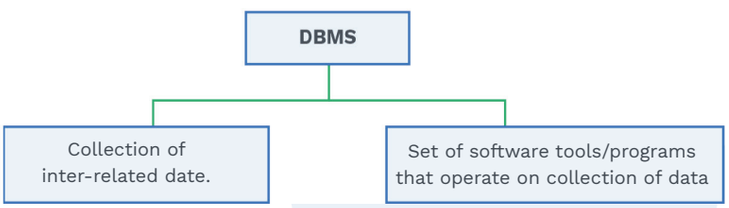
1 Introduction to DBMS

1. INTRODUCTION TO DBMS

A system which manages all such inter-related data is called “Database Management System”. It plays a vital role in almost all areas where computers are used.

For example, a company can have a database of their employees, including their name, address, age, date of birth and salaries.

DBMS:



Database management system can be defined as software that helps to maintain and utilize a huge amount of data.

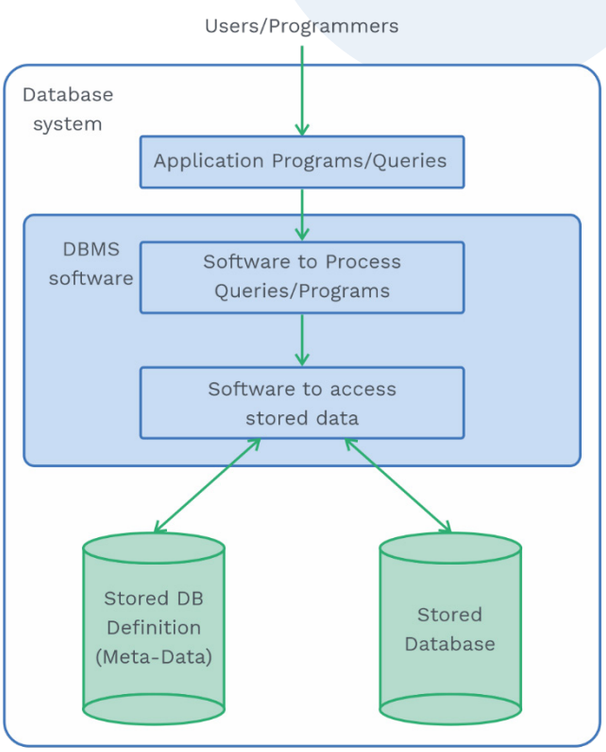


Fig. 1.1 A Simplified Database System Environment

DBMS helps in:

1) **Database design:** It helps to decide how to organise the stored data.
2) **Data analysis:** It tells to execute queries over data in DBMS.
3) **Concurrency and robustness:** DBMS allows multiple users to access data concurrently and protects data in case of system failures.
4) **Efficiency and scalability:** DBMS efficiently give answers to the queries and the cost-effectiveness, ability to accommodate increased workload and makes it scalable.

File systems versus a DBMS:

To store a large amount of data, we need a database management system. For example, a company has a large amount of data on employees, departments etc. In this case, only DBMS can give us the results, not file systems.

Let us consider an example: A company has a large amount of data, say 500 GB, on employees. Now, this data needs to be accessed concurrently by several employees, changes can be made to a certain parts of data, and these should be applied consistently and access to certain part of data must be restricted.

Using operating system files to store this huge amount of data, this may cause certain drawbacks.

- It is possible that we may not have 500 GB of main memory to hold all the data.
- We must write special programs to answer each question.
- Operating systems are inflexible to enforce security.

We can store data using DBMS instead of files to manage the data efficiently.

File System	DBMS
Data is stored on the disk	Data is being stored by database management system using indexing
Accessing any information is time consuming	Time required to access is comparatively less.
It can have redundant data	DBMS reduces redundancy

Table 1.1 File System Vs DBMS

Note:
DBMS stores everything in a file, but DBMS also includes a piece of software that helps manage these data efficiently.

The usefulness of different representations of the database:

i) We know how entities are related. Therefore, the databases can be pictorially represented using E-R diagrams.

ii) We need tables to store data. The model which talks about it is called the relational model.

iii) Searching and retrieving is fast. Therefore, we use B-Trees.

iv) The accessing of data can be made simple using transaction and concurrency control system.

Architecture of database:

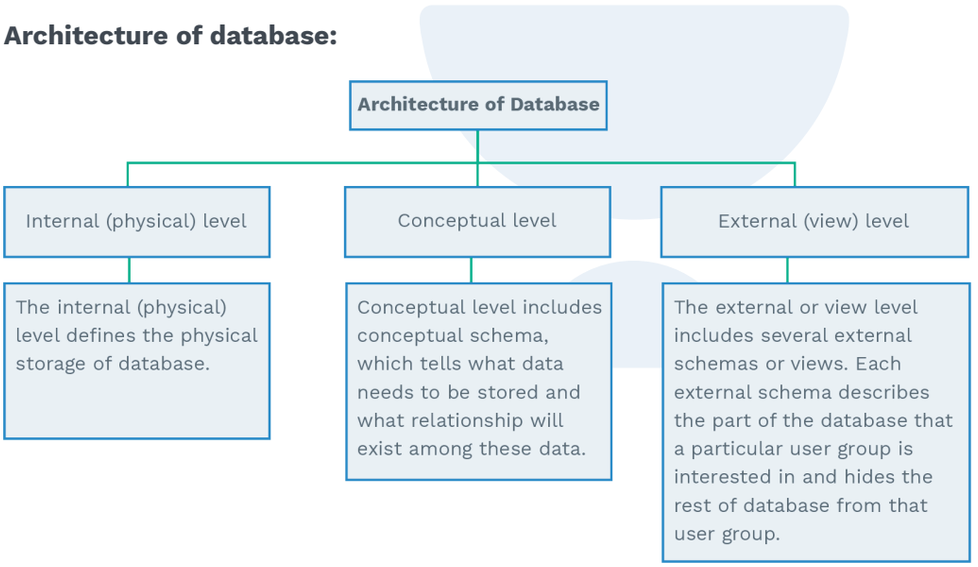


Fig. 1.2 Architecture of Database

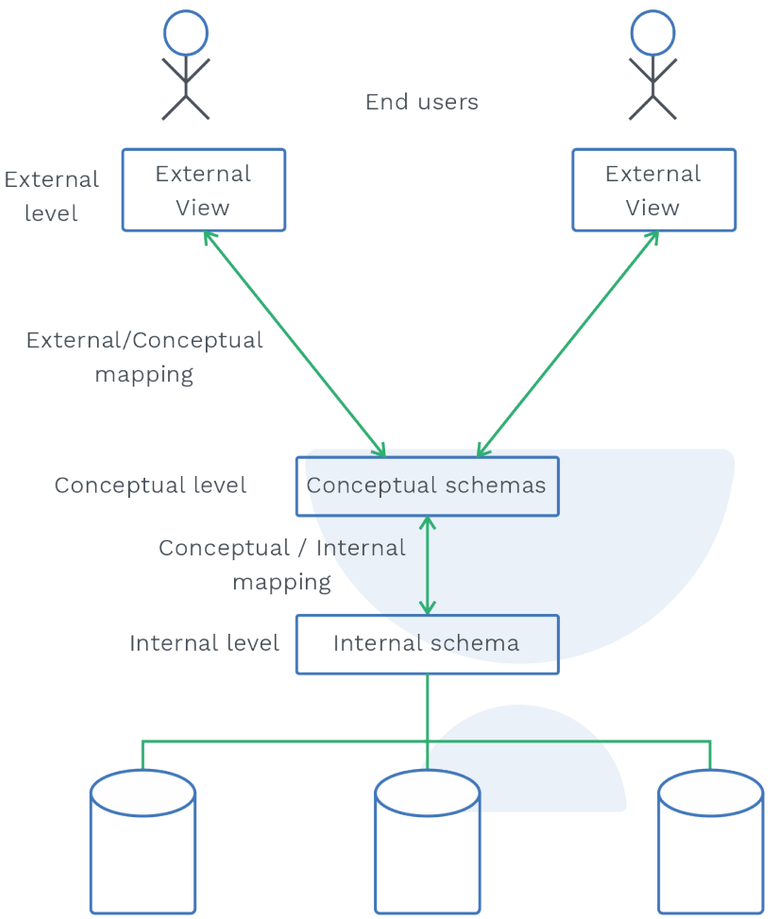


Fig. 1.3 Stored Database

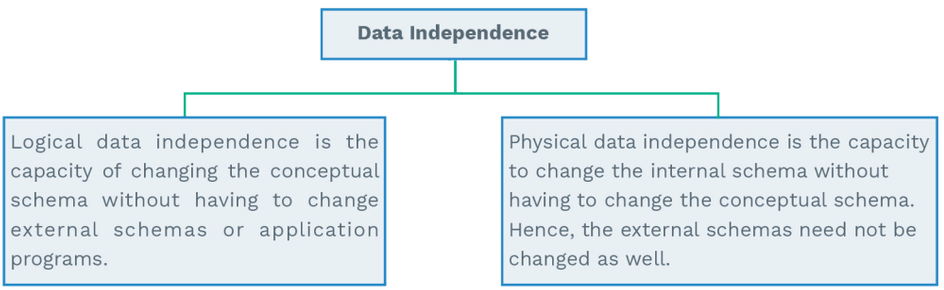


Fig. 1.4 Data Independence

Advantages of DBMS:

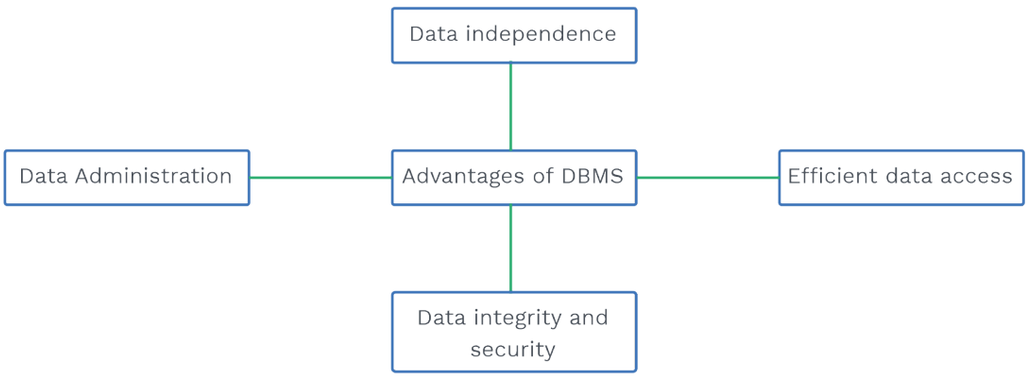


Fig. 1.5 Advantage of DBMS

Concept of keys and constraints:

Keys

An attribute or a set of attributes which helps in uniquely identifying a tuple (a row) in a relation (a table) is called a key.

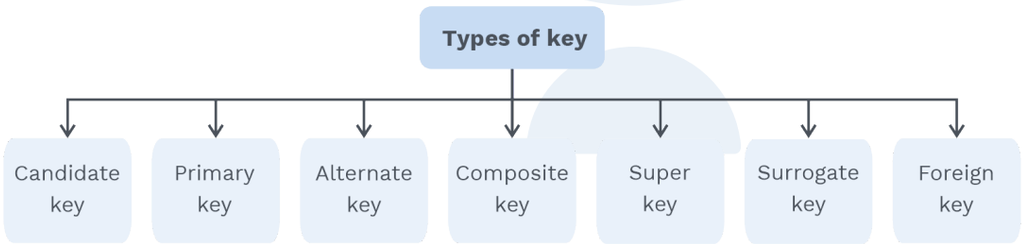


Fig. 1.6 Types of Keys

Compound key: Key with multiple attributes/columns.

Example: (emp_name, emp_dept) is a compound key.

Candidate key: The set of all unique keys are candidate keys.

Example: {emp-id, {emp-name, emp-dept}}

Rules while closing primary key:

- It can not be null of any tuple/row.
- There should be at most one primary key per table.
- It should be unique for each row/tuple.

Above mentioned conditions are called entity integrity constraints. Constraints are properties to be satisfied while inserting, deleting, or modifying the data in a relational table.

- Foreign key: It is an attribute/group of attributes in a relation database which relates two tables.

Example: Consider following two tables.

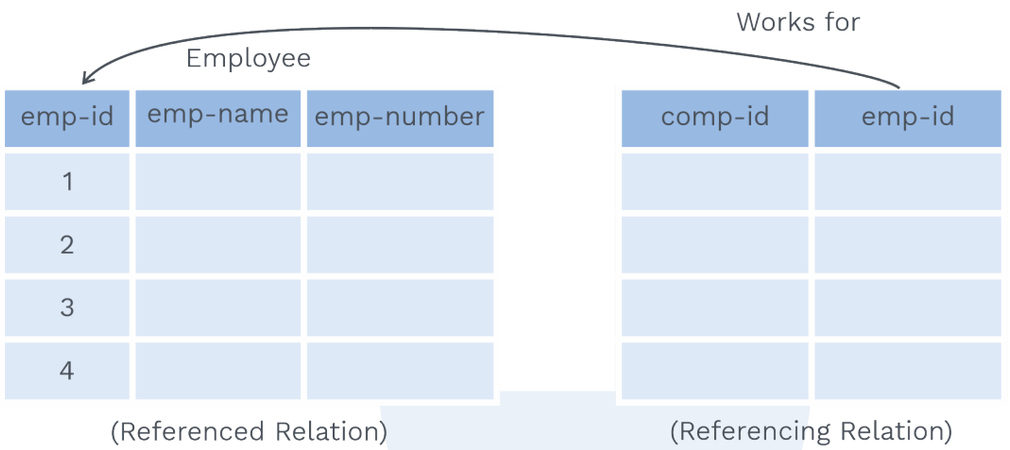


Table 1.2 Foreign Key

'Emp-id' is a foreign key in Works for relation. A foreign key should be the primary key in referenced relation, but it may or may not be the primary key in referencing relation.

Self-referential foreign keys: It defines relationships within the same table.
Example:

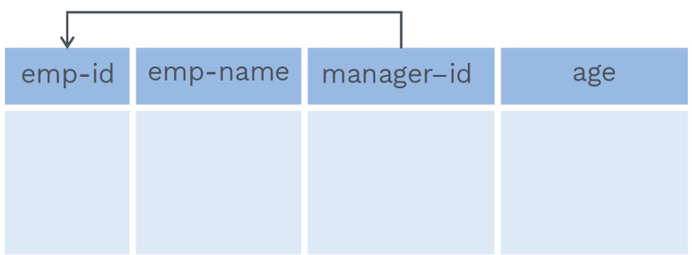


Table 1.3 Self-referential Foreign Keys

Here, the emp-dept id is a self-referential foreign key.

Surrogate key: It is an artificial key which uniquely identifies each record.
Super-key: It's a collection of one or more attributes that allow us to identify an entity in the relation uniquely. The superset of the candidate key is known as the super key in a relation.
Example: consider Emp-id as a candidate key in a relation Employee. Then, {Emp-id, Emp-name} is a super-key.

Q1 **How many super keys are possible in R (A_1, A_2 ... A_n) when candidate key is {A_1}?**

Sol: Super key = ((candidate key) ∪ (other attributes))

Here, candidate key = A_1

Now, $A_1 \cup \phi, A_1 \cup A_2, \ldots A_1 \cup A_2, A_3 \ldots A_n$ all are possible super keys. It means A_1 can be combined with (n-1) ways to other attributes in a given relation.

Using the concept of power sets:

Number of super keys = 2^{n-1}.

Q2 **How many super keys are possible in relation R (A_1, A_2, ... A_n) if candidate keys are {A_1, A_2}?**

Sol: As we have discussed in the above example

- Taking A_1 as a candidate key.
  Number of super keys = 2^{n-1}, say A_1
- Taking A_2 as a candidate key.
  Number of super keys = 2^{n-1}, say A_2
  Using inclusion-exclusion principal

$$(|A_1| \cup |A_2|) = |A_1| + |A_2| - |A_1| \cap |A_2| = 2^{n-1} + 2^{n-1} - 2^{n-2} = 2^n - 2^{n-2}.$$

Rack Your Brain

How many super keys will be in a relation R (A_1, A_2, ... A_n) when the candidate keys are {{A_1, A_2}, {A_3, A_4}}

Rack Your Brain

Consider R (A, B, C, D); how many superkeys are possible if

i) A is the candidate key.

ii) {A, B, C} are candidate keys.

Candidate key: The minimal superkey is defined as the candidate key.
Primary key: Randomly chosen candidate key is the primary key.

Let us consider two tables, student and subset, with mentioned attributes.

Student				
S–id	S-name	S-acc no	S-roll no	Sub id

Subjects	
Sub-id	Sub-name

Table 1.4 Tables

Here, Candidate keys = {S-id, S-acc no, S- roll no}
Primary key = {S-id}
Alternate keys = {S-acc no, S-roll no}
Super keys = {S-id, {S-id, S-name}, {S-id, S-name, S-roll no}...}
Foreign key = {Sub-id}

Alternate Key: All other keys are alternate keys from the set of candidate keys, except the primary key.

Constraints:
As we have already discussed, constraints are conditions which must be satisfied before performing any operation on the database.

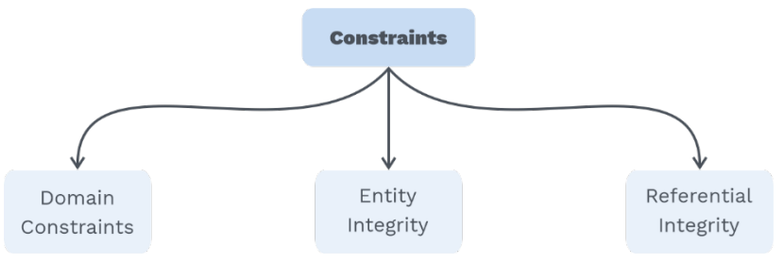


Fig. 1.7 Constraints

i) **Entity integrity constraints:**
- Every relation must have a primary key
- Every primary key must be unique
- A primary key should not be NULL

ii) Domain integrity constraints:

Domain integrity constraints just give a valid value for the attribute.

Example: An employee's age must be between 24 and 35, and the domain should be (integer) numeric value.

Referential integrity constraints:

- It is based on a foreign key concept.

Example:

Employee

emp-id	emp-name	dept-id
1	Jannat	101
2	Faisu	102
3	Muskan	103

Table 1.5 Referencing Relation

Department

dept-id	dept-name
101	Acting
102	Singing

Table 1.6 Referenced Relation

Now, various operations can be performed on referencing and referenced relations.

The three basic operations are:

- Insertion
- Deletion
- Updation

Referenced Relation	Referencing Relation
On insertion	On inserting
On inserting in referenced relation, there will be no change in referencing relation.	On inserting in referencing relation, first it will check if it is not violating referential integrity constraint.
On deletion	On deletion
On deleting a row, changes will be made: **i)** On delete no action **ii)** On delete cascade. This will be a deletion of a row from referencing relation corresponds to particular key. **iii)** On delete set null: It will set value null corresponding referencing tuple.	On deletion in referencing relation, it will bring no change.
On updation	On updation
i) On updation no action. **ii)** On updation cascade It will update the changes in referencing relation. **iii)** On update set null is not widely used.	It will also check for violation.

Table 1.7 Referenced Vs Referencing Relation

Note:

- On delete cascade can result in loss of data.
- On delete, no action violates referential integrity.

Relational model:

The database is essentially represented as a collection of relations in the relational paradigm. E.F Codd proposed it to model data in the form of relations or tables.

Example:

emp-id	Name	Address	Contact	Dept	Age
1	Sushant	Mumbai	9213213400	CS/IT	27
2	Rhea	Bhopal	9132123400	Civil	31
3	Showick	West Bengal	9123912300	Mechanical	24
4	Kangana	Himachal	9989998900		37
5	Diljit	Punjab	9789978900	Electrical	29
6	Arnab	Delhi	8760876000	Communication	42

Table 1.8 Relational Model

Terminology:

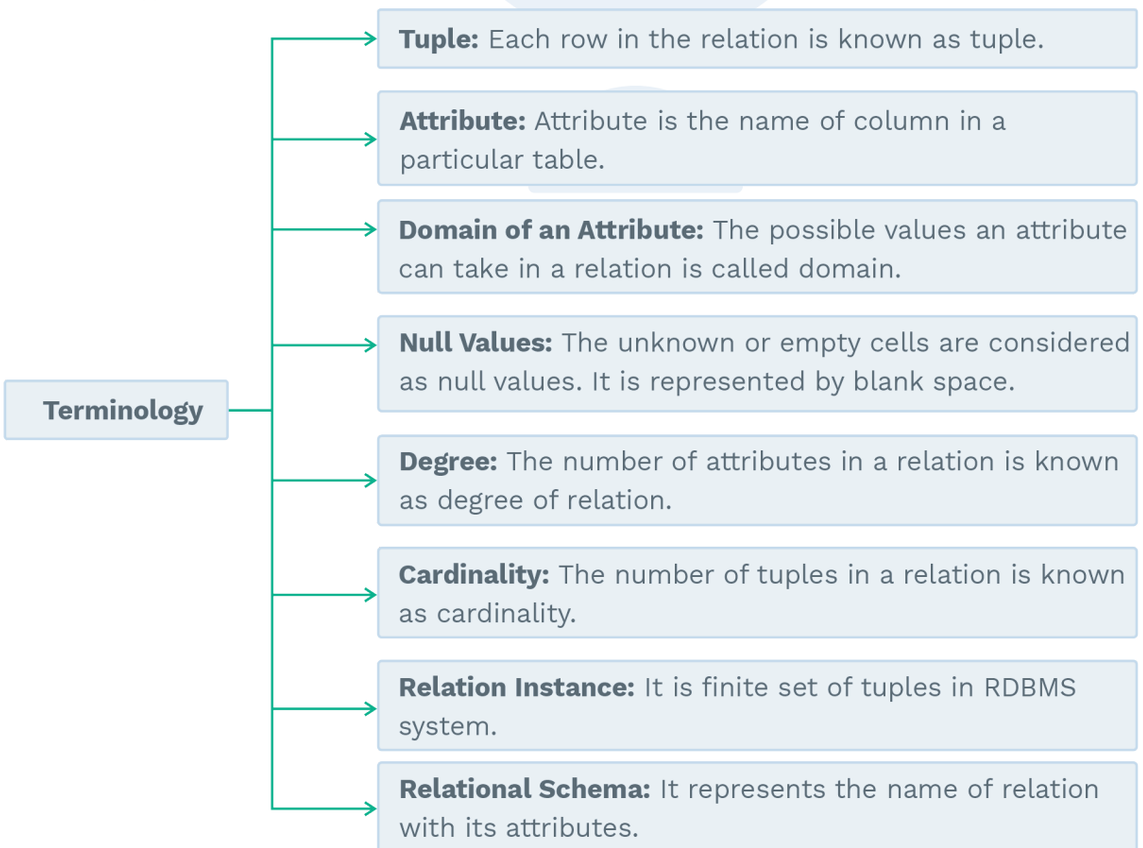


Fig. 1.8 Terminology

Note:

Relation instances never have duplicate tuples.

Example: Employee

emp-id	Name	Address	Contact	Dept	Age
1	Sushant	Mumbai	9213213400	CS/IT	27
2	Rhea	Bhopal	9132123400	Civil	31
3	Showick	West Bengal	9123912300	Mechanical	24
4	Kangana	Himachal	9989998900		37
5	Diljit	Punjab	9789978900	Electrical	29
6	Arnab	Delhi	8760876000	Communication	42

Table 1.9 Employee Data

Tuple:	1	Sushant	Mumbai	9213213400	CS/IT	27

Attributes: emp-id, Name, Address, Contact, Dept, Age.

The domain of attribute Age: The age range should be greater than 20 and less than 45.

Null values: emp-id = 4 has no dept assigned.

Cardinality: 6

Codd presents his 13 rules for a database to test the concept of DBMS against his relational model, and if a database follows the rule, it is called RDBMS. There are 13 popular rules known as Codd's 12 rules:

Fig. 1.9 ER Model

The E-R model consists of three basic terms:

- Entity sets
- Relationship sets
- Attributes

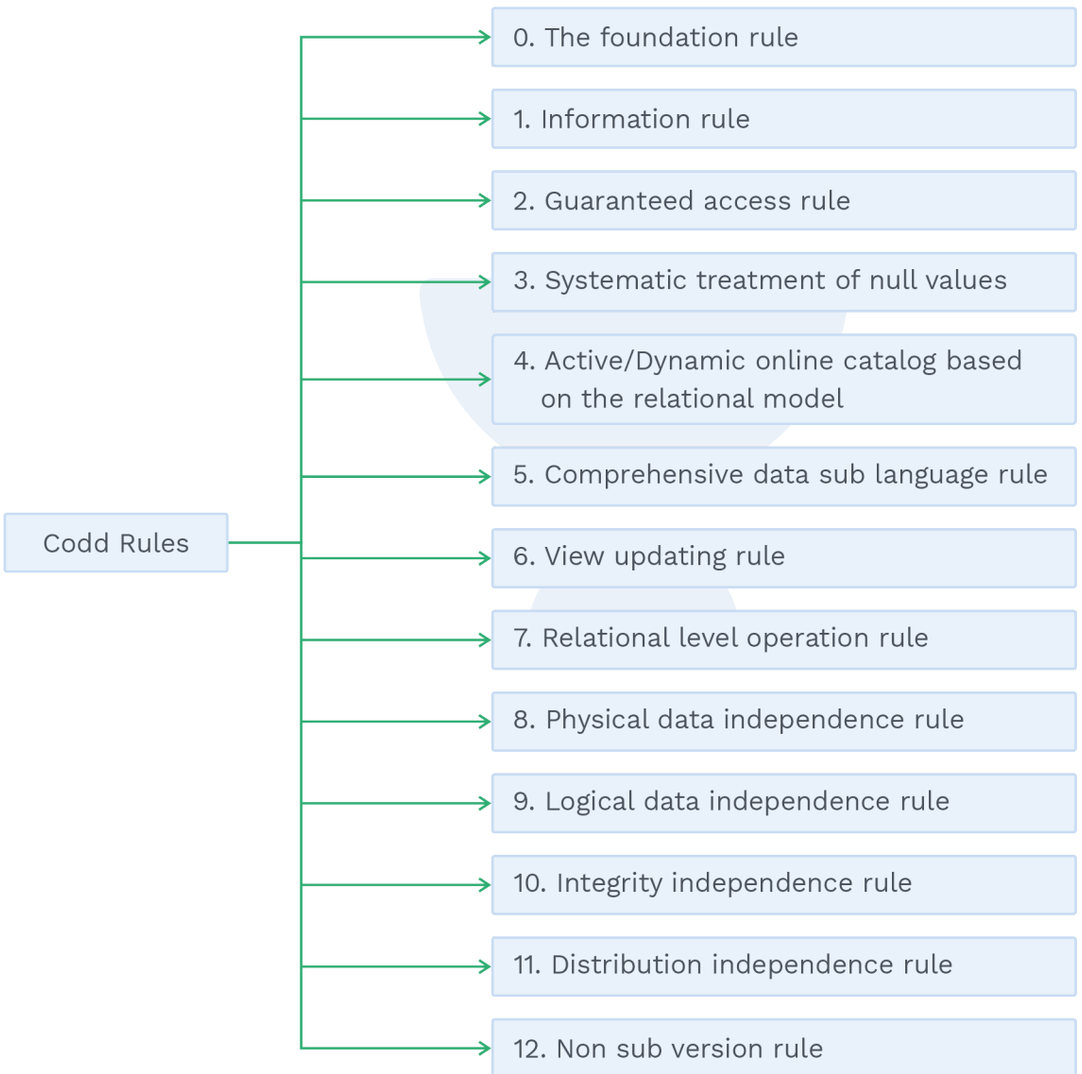


Fig. 1.10 Codd Rules

Entity sets:

An entity is a distinct object that can be differentiated from other items. It has a set of properties, some of which can be used to identify an entity. For example, an employee will have an id which uniquely identifies one

particular employee. In addition, an entity can be concrete (person, book) or abstract (loan, holiday or concept).

A set of entities of the same type with the same features or attributes is referred to as an entity set. Individual entities that make up a set are referred to as the extension of the entity set in a database management system (DBMS). All individual bank clients, for example, are an extension of the entity set customer.

It is not required for entity sets to be disjoint. A set of attributes represents an entity. Each of the attributes of an entity has a value. For example, empid, emp-name, and emp–address may have value for a certain employee object.

Fig. 1.11 Types of Entity Set

Strong entity set:
It has enough properties to identify each of its entities uniquely. So, formally we can say that a strong entity set has a main key.
A rectangle represents it.

Example:

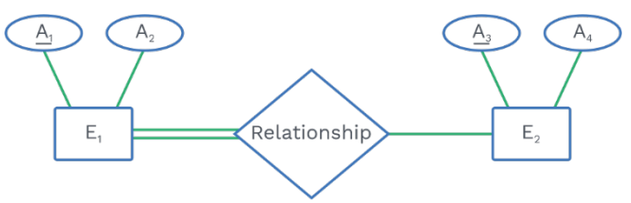


Fig. 1.12 Strong Entity Set

Where, E = Entity set
A = Attribute

Weak entity set:
It lacks sufficient properties to allow its entities to be uniquely identified. Although the weak entity set does not have a primary key, it does have a partial key, known as a discriminator, that can identify a group of entities from the entity set. A dashed line is used to depict it.
Weak entity sets are represented by a double rectangle, while the relationship between strong and weak entity sets is represented by a double diamond symbol. This relationship is known as the identifying relationship.

Note:
Total participation exists in a relationship involving weak entities, from the weak entity terminal.

Example:

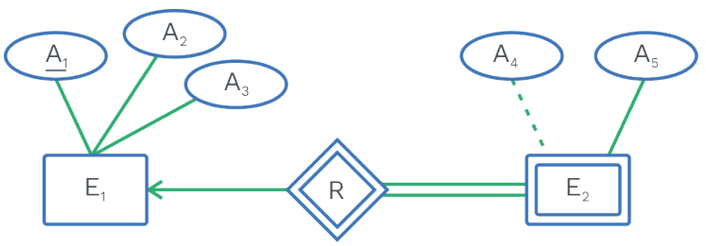


Fig. 1.13 Weak Entity Set

Where, E_2 is a weak entity set
E_1 is a strong entity set
A_4 is the discriminator of a weak entity set E_2

Relationship sets:
A connection is essentially an association between two or more entities. A relationship set is a collection of similar relationships.

The association between entity sets is referred to as participation. The entity's function in a relationship is called that entity's role.

Note:
A relationship may also have attributes called descriptive attributes.

The number of entity sets that participates in a relationship set is called the degree of a relationship set. A binary relationship set is degree 2, a ternary relationship set is degree 3.

Attributes:
Attributes can be of the following types:

1) **Simple and composite attributes:** The attributes which cannot be decomposed into a set of smaller independent attributes are classified as simple attributes. Composite attributes can be further split into more than one simple attribute.

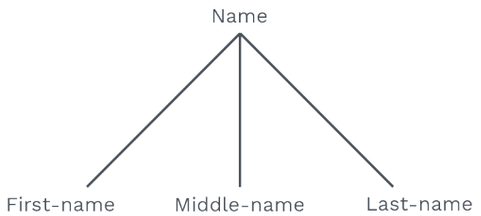


Fig. 1.14 Composite Attributes Employee Name

2) **Single-valued and multivalued attributes:** Single valued characteristics are attributes that have a single value for a specific entity. A multivalued attribute is one that has a collection of values for a certain entity.
3) **Derived attribute:** An attribute whose field can be derived from other attributes is classified as a derived attribute.
   Consider the following scenario: The date of birth can be used to calculate age.

Constraints:

There are two major types of constraints which we will discuss:

- Mapping cardinalities
- Participation constraints

i) **Mapping cardinalities:** Mapping cardinality denotes the number of associativity from one entity set to another. The following forms of mapping cardinalities for a binary relationship set R on entity sets A and B are possible:

1) **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.

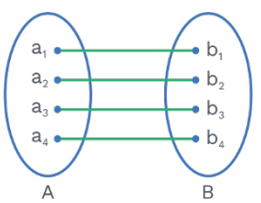


Fig. 1.15 One-to-one Mapping

2) **One-to-many:** A single entity in set A can be linked to any number of entities in set B, but one entity in set B can be linked to at most one entity in set A.

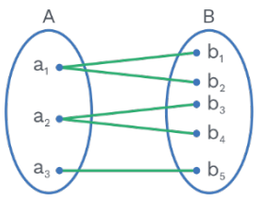


Fig. 1.16 One-To-Many Mapping

3) **Many-to-one:** Every component in set A must have at most one correspondence in set B. Every component in set B may have zero or more correspondences in set A.

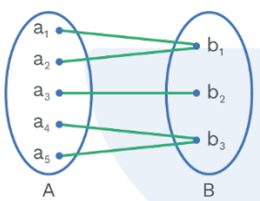


Fig. 1.17 Many-to-one Mapping

4) **Many-to-many:** Every component in set A may have any number of correspondences in set B and conversely.

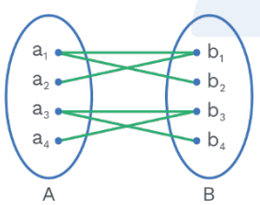


Fig. 1.18 Many-to-many Mapping

ii) Participation constraints:

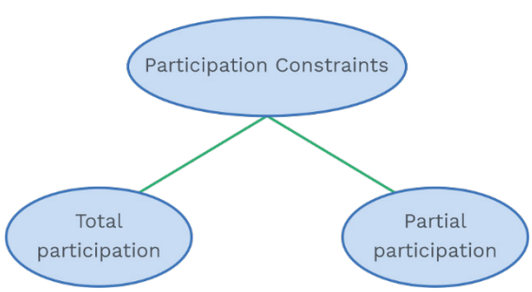
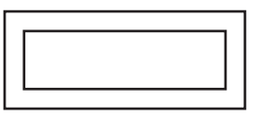


Fig. 1.19 Participation Constraints

If every entity in E participates in at least one relationship in R, the participation is said to be total.
However, if just some entities participate in relationship R, entity set E's participation is considered partial.

Relationship sets:
Let R be a set of relationships involving entity sets E_1, E_2,... E_n. Let the primary key (E_i) signify the set of attributes that make up the entity set Ei's primary key.

Entity-relationship diagram:
ER models consist of the following major components.

- Rectangle  Represent entity set
- Eclipse  Represent attributes
- Diamonds  represent relationship sets
- Lines Attributes are linked to entity sets, while entity sets are linked to relationship sets.
- Double eclipse  represent Multivalued attributes
- Dash oval  represents derived attributes
- Double lines Total participation
- Double rectangles  Weak entity sets

Example of ER diagram:

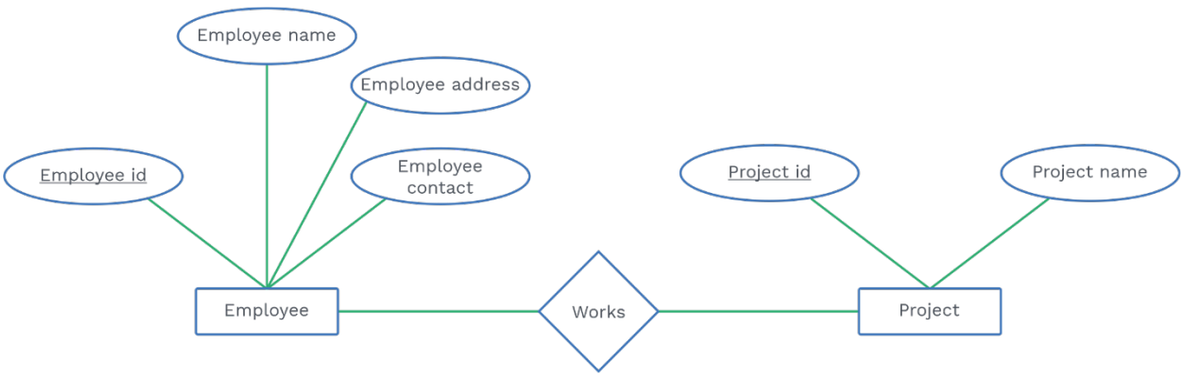


Fig. 1.20 ER Diagram

Here in the above diagram:

- Underlined entity is the primary key.
- An undirected line represents a many-to-many relationship.
- A directed line may represent a one-to-many or many-to-one relationship.

Example 1:

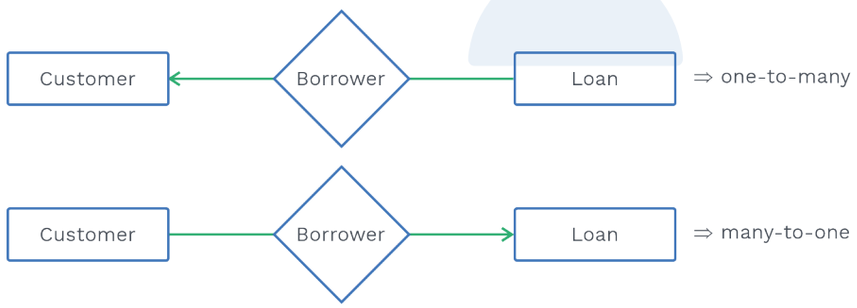


Fig. 1.21 Example

Example 2: The following figure shows ER diagram with composite, multivalued and derived attributes.

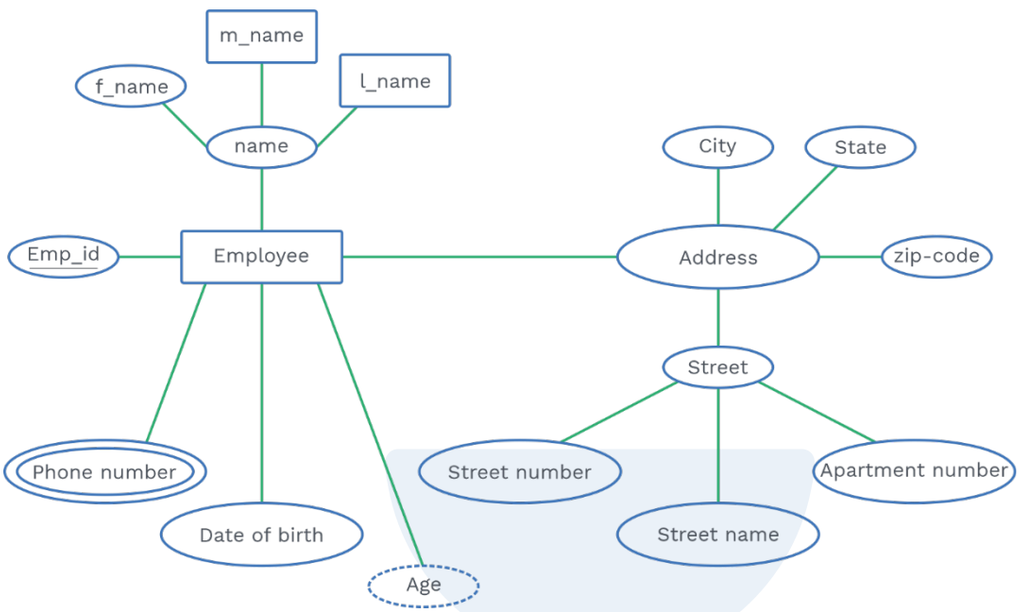


Fig. 1.22 ER Diagram

Minimization of ER diagrams:

We can minimize ER diagrams into tables because RDMS can easily organise tables.

To minimize ER diagrams, following rules should be kept under consideration.

I) A strong entity with only simple attribute will require only one table.

- Attributes of the entity set will be the attributes of table.
- Primary attributes of the entity set will be primary key of table.

For example:

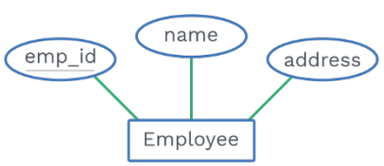


Fig. 1.23 ER Diagram

⇒

emp_id	name	address

II) Check for strong entity set with composite attributes

- A strong entity with any number of composite attributes will require only one table.

Example:

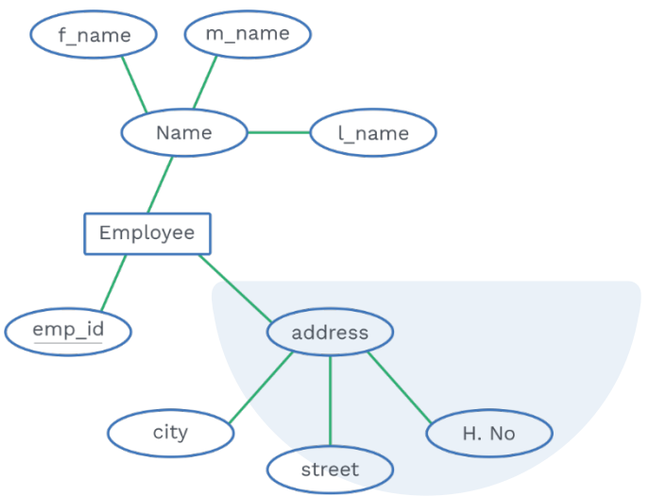


Fig. 1.24 ER Diagram

⇒

emp_id	f_name	m_name	l_name	City	Street	H.No.

III) Check for strong entity set with multivalued attributes.
This requires two tables:

- One table will contain all simple attributes with primary key.
- Other table will contain primary key all the multivalued attributes.

Example:

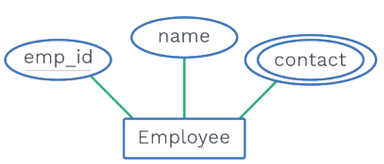


Fig. 1.25 ER Diagram

Table 1:

emp_id	name

Table 2:

emp_id	contact

IV) Translation of relationship set into a table:
It requires one table in relational model and its attributes should be:

- Primary key attributes of entity sets
- Descriptive attributes if any

Example:

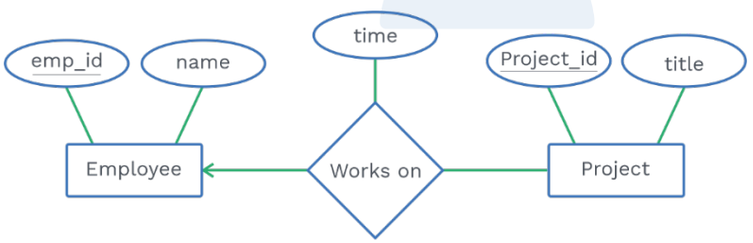


Fig. 1.26 ER Diagram

emp_id	proj_id	time

This design requires three relations- Employee, Project, Works on.

V) Check for binary relationships with cardinality ratios.

Case I:
Many-to-many
Example:

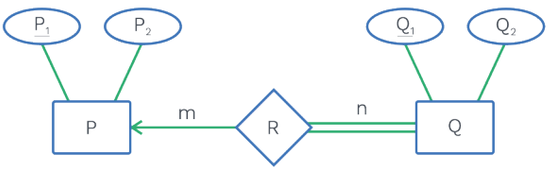


Fig. 1.27 Many-to-one Example

Here, three tables will be required:

- P (P_1, P_2)
- Q (P_1, Q_1)
- R (Q_1, Q_2)

Case II: One-to-many:

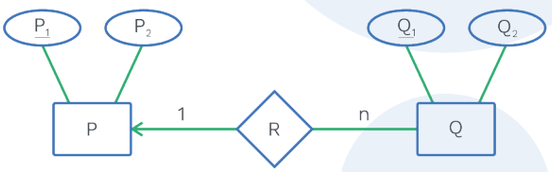


Fig. 1.28 One-to-many Example

Here, two tables will be required

- QR (P_1, Q_1, Q_2)
- P (P_1, P_2)

Case III: Many-to-one:

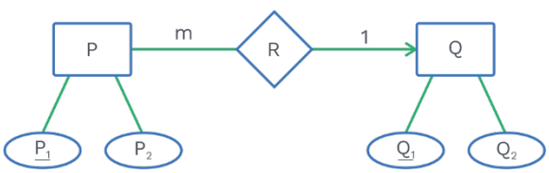


Fig. 1.29 Many-to-one Example

Here, two tables will be required:

- PR (P_1, P_2, Q_1)
- Q (Q_1, Q_2)

Case IV: One-to-one:

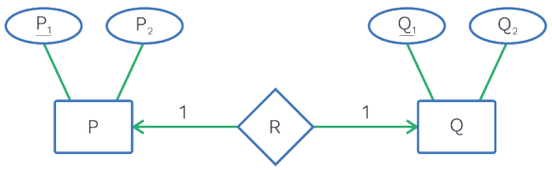


Fig. 1.30 One-To-One Example

In this case, there are two ways

VI) Check for the binary relationship between cardinality and participation constraints.

Case I: On one side, there is a binary relationship between a cardinality constraint and a total participation constraint.

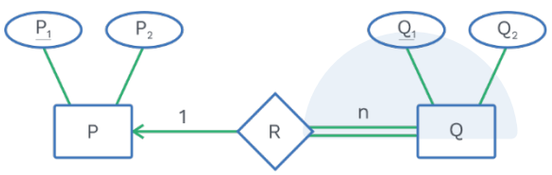


Fig. 1.32 ER Diagram

As we have already discussed, for one-to-many, two tables will be required

- P (P_1, P_2)
- QR (Q_1, P_1, Q_2)

But, because of total participation, the foreign key acquires Not NULL constraint, i.e., the new foreign key cannot be null.

Case II: Binary relationship with cardinality constraint and total participation constraint from both sides.

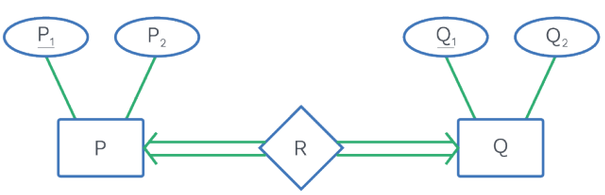


Fig. 1.33 Binary Relationship Diagram

Please keep in mind that if both sides of an entity set have a key constraint with total participation, the binary connection is represented using only one table.
Therefore,
PRQ is the table that will generate in this case (P_1, P_2, Q_1, Q_2)

VII) With a weak entity set, view for binary relationships. Please remember that a weak entity set is always associated with an identifying connection that has a total participation constraint.

Example:

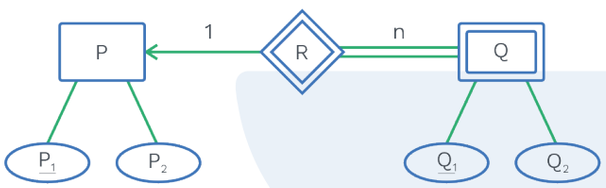


Fig. 1.34 ER Diagram

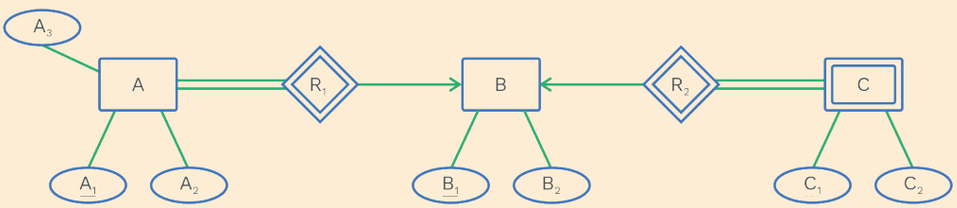
Therefore, here two tables will be required:

- P (P_1, P_2)
- QR (P_1, Q_1, Q_2)

The above mentioned seven rules are the basic and main rules used in reducing of E-R diagrams into tables.

SOLVED EXAMPLES

Q3 **Find the minimum number of tables required for the following ER diagram in relational model.**



Sol: With the help of the above discussed rules, we can conclude that a minimum of 3 tables will be required.

Table 1:

AR_1 =

A_1	A_2	A_3	B_1

Table 2:

B =

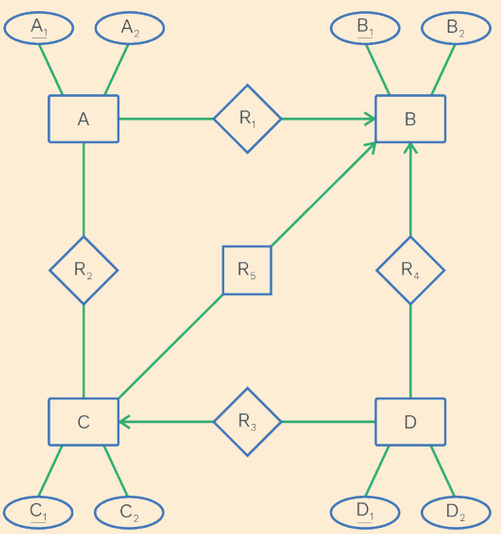
B_1	B_2

Table 3:

CR_2 =

B_1C_1	C_1	C_2

Q4 **Consider the below ER model:**



How many are the minimum number of relations needed to organise the above ER model into RDBMS design?

Sol: Considering the above given ER diagram and applying rules, minimum 5 tables will be required.

Table 1:

AR_1

A_1	A_2	B_1

Table 2:

B

B_1	B_2

Table 3:

CR_5

C_1	C_2	B_1

Table 4:

DR_3R_4

D_1	D_2	C_1	B_1

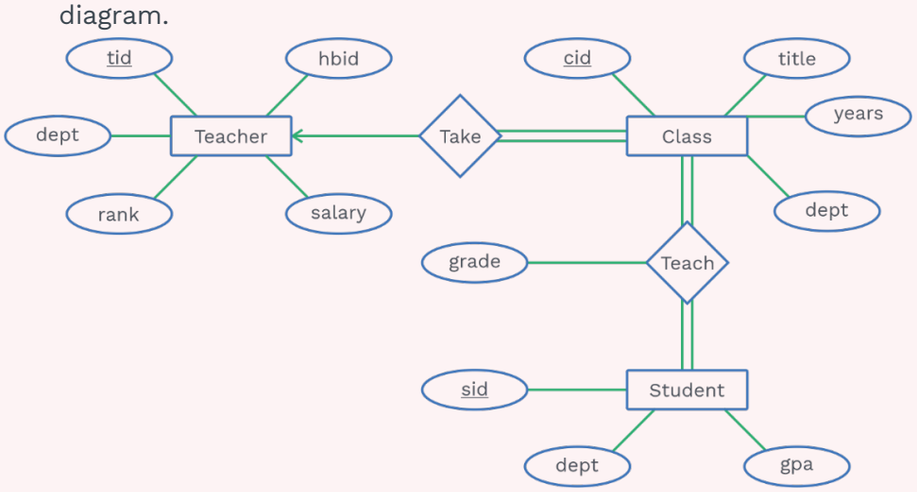
Table 5:

R_2 =

A_1	C_1

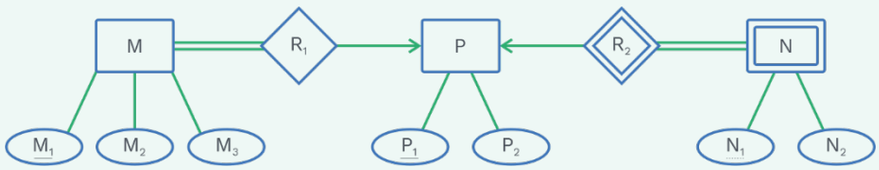
Rack Your Brain

Q. Find the minimum number of tables required for the given ER diagram.



Previous Years' Question

Consider the following ER diagram.



Which of the following is a correct attribute set for one of the tables for the minimum number of tables needed to represent M, N, P, R_1, R_2?

1) $\{M_1, M_2, M_3, P_1\}$ **2)** $\{M_1, P_1, N_1, N_2\}$

3) $\{M_1, P_1, N_1\}$ **4)** $\{M_1, P_1\}$

Sol: 1) **(GATE-CSE 2008)**

Self-referential relationship:

It is the relation from an entity to itself. For example, a manager being an employee managing another employee.

General diagram of show the self-referential relationship:

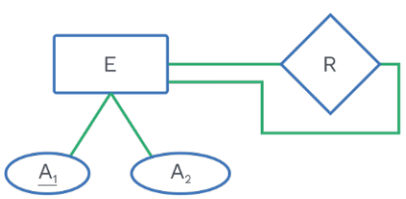


Fig. 1.35 Self-Referential Relationship

Case I: One-to-one relationship:

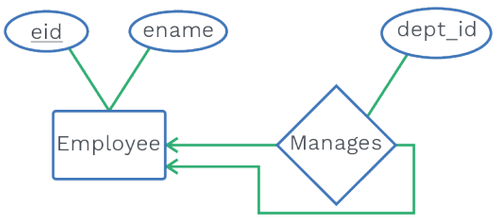


Fig. 1.36 One-to-one Relationship

Only one table will be required.

Case 2: One-to-many relationship

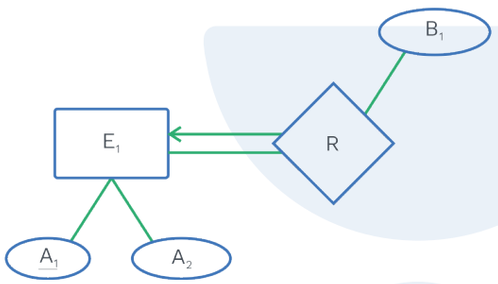


Fig. 1.37 One-to-many Relationship

Only one table will sufficient.

Case 3: Many-to-many relationship:

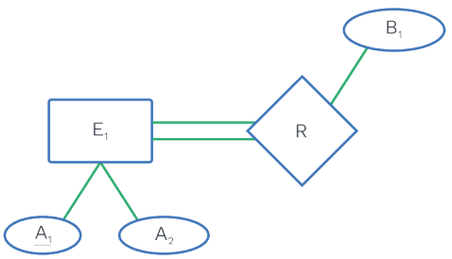


Fig. 1.38 Many-to-many Relationship

Two table will be required.

Extended ER-features:

Specialization:

- Specialization refers to the presence of more detailed attribute set belonging to an entity that uniquely defers it from other entities.
- Consider a person entity set with attributes such as name, street, and city. A person can also be described as one of the following:

- Customer & Employee
- Specialization is the process of identifying subgroupings within an entity set. We can discriminate between employees and customers using person specialization.
- Specialization is represented in the E-R diagram by the triangle component labelled ISA, which stands for “is-a” and symbolises, for example, a client is a person. A superclass-subclass relationship is another name for this relationship.

Generalization:

Generalization refers to the similarity of attributes shared between two or more entities.

For all the practical purposes, generalization is simple inversion of specialization. “Difference in the two approaches may be characterized by their starting point and our all goal.

Specialization stems from a single entity set, it emphasises difference among entities within the set by creating distinct lower-level entity sets. Generalization proceeds from the recognition that a number of entity sets share some common features.”

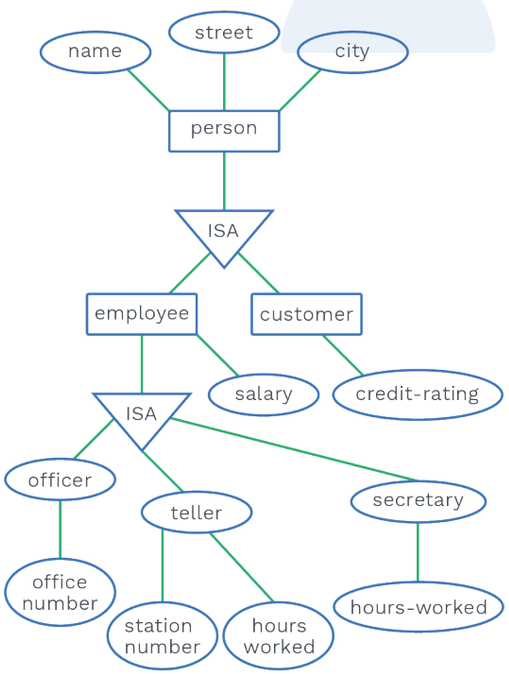


Fig. 1.39 Specialization and Generlization

Function dependency

In this topic we are going to study the following things

- Functional Dependencies
- Types of functional dependency
- Rules of functional dependency
- Attribute closure
- Minimal cover
- Problem caused by redundancy
- Equivalence set of functional dependency

Here, we will concentrate on an important class of constraints called functional dependencies.

Definition

Functional dependency (FD) is a kind of integrity constraint that generalizes the concept of a key.

Let R be a relation and let X & Y be non-empty sets of attributes in R
If the following holds for every Tuple t_1 and t_2 in r, we claim that an instance r of R fulfils the FD X ® Y.

If $t_1 \times X = t_2 \times X$ then $t_1 \times Y = t_2 \times y$.

Where t_1 and t_2 are two difference tuples.

Consider an entity set employee with attributes emp-id, emp-name, empaddress.

Employee (emp-id, emp-name, emp-address)
Now,

1) If given an employee id, we can uniquely determine the employee's name. It can be represented as:
emp-id → emp-name ...(i)

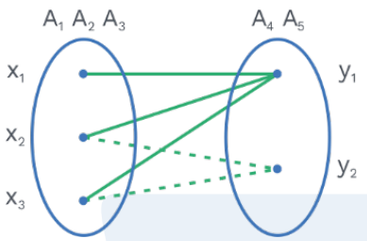
2) Similarly, given an id, we can uniquely determine the address
Therefore, emp-id → emp-address ...(ii)
Now, combining (i) & (ii) we get
emp-id → emp-name, emp-address

Note:

'→' is a notation which means LHS functionally determines RHS.

Generalization, given a relation $R(A_1, A_2, ..., A_n)$
Let $\{A_1, A_2, A_3\} \rightarrow \{A_4, A_5\}$

Now, let's try to understand this concept using set theory.



Types of functional dependencies:

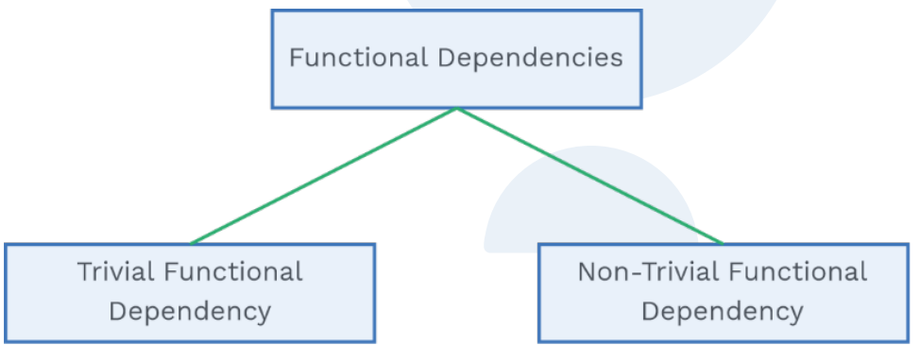


Fig. 1.40 Types of Functional Dependencies

- Any Functional dependency is said to be Trivial iff LHS is a superset of RHS.
  Considering the above example:
  emp-id → emp-id
  emp-id, emp-name → emp-id
  emp-id, emp-name → emp-name
- Any Functional dependency is said to be non-trivial if LHS is not a superset of RHS :
  For example, emp-id → emp-name

Rules of functional dependency:

Over a relation schema R, we employ X, Y, and Z to denote sets of attributes:

1. Reflexivity: If $X \supseteq Y$, then $X \rightarrow Y$

2. Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z

3. Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

There are some additional rules as well.

1) Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

2) Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Attribute closure of functional dependency:

A collection of all Functional dependencies implied by a particular set F of Functional Dependency is defined as the closure of F. To check if a given Functional dependency is in the closure of a set F of FDs, we can do this using the following algorithm.

Closure = X
Repeat until there is no change: {
If there is an FD: $U \rightarrow V$ in F such that
$U \subseteq$ closure,
then set closure = closure UV
}

Attribute closure of X: It is a set of attributes that can be functionally determined using attributes in X.

Example 1: Given relation R(A, B, C) and FD = $A \rightarrow B$, $B \rightarrow C$
Now, let's compute the closure of A (A^+), which is a set of all attributes which can be determined using A.

$$A^+ = \{A, B, C\}$$

Similarly:

$$B^+ = \{B, C\}$$

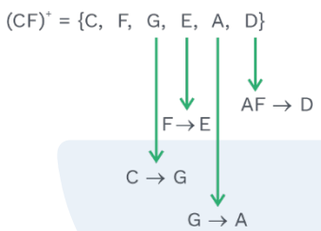
$$C^+ = \{C\}$$

Example 2: R(A, B, C, D, E, F, G) and set of FD =
{
$AB \rightarrow CD$

AF → D
DE → F
C → G
F → E
G → A
}

Find closure of CF?

Sol:



Closure of attributes helps us to determine super keys and candidate keys.

- If F^+ gives all attributes of R, then F is super key.

Note:
Attributes which are part of atleast one candidate key are called prime attribute, non-prime otherwise, and the concept of prime attribute helps us to determine all possible candidate keys.

SOLVED EXAMPLES

Q5 **Consider a relation R with the attributes (P, Q, M, S, T, U, V, W) with the following functional dependencies :**
P → Q, S → V, TU → W, MS → T, Q → U, U → S, S → P, T → M. Which of the following is/are keys for relation R?
1) MS 2) STU 3) PS 4) PU

Sol: **1) and 2)**

1) MS → T, S → V, S → P, P → Q, Q → U, TU → W.
(MS)+ = {P, Q, M, S, T, U, V, W}
∴ MS is a key for R.

2) (STU)+ = {S, T, U, V, W, P, M, Q}
∴ STU is a key for R.

3) (PS)+ = {P, S, Q, V, U}
∴ PS is not a key for R.
4) (PU)+ = {P, U, Q, S, V}
∴ PU is not a key for R.

Q6 **R(ABCDE)**
FD = {A → BCDE,
BC → ADE,
D → E}
Find candidate key?

Sol: $A^+ = \{A, B, C, D, E\}$
Now, $B^+ = \{B\}$
$C^+ = \{C\}$
$D^+ = \{D\}$
$E^+ = \{E\}$
But BC → A, now we will check for BC,
$(BC)^+ = \{B, C, A, D, E\}$
Therefore, BC is a candidate key.

Properties of functional dependency set:

1) Membership: Let F be an FD set on R we can get X → Y using FD in F then, X → Y is a member of F.

Example: R{ABC} and FD = {P → Q, Q → R}
Is P → R is member of F?
Sol: $P^+ = \{P, Q, R\}$
Using closure of P, we can say that

$$A \rightarrow C$$

Therefore, P → R is a member of F.

2) Closure of an FD set: It is a set of all functional dependencies that can be determined using Functional dependency in F.

Note:
F^+ includes all trivial and non-trivial dependencies that are possible.

3) **Equality of functional dependency set:** Two Functional dependency, F and G, is said to be equal iff are
   - $F^+ = G^+$
   - F covers G, and G covers F (If all FDs in G can be determined by all FD's in F and all FDs in F can be determined by all FDs in G).

SOLVED EXAMPLES

Q5 **For R(PQRS), given** **F = {PQ → RS, Q → R, R → S}**
G = {PQ → R, PQ → S, R → S}

Is F = G?

Sol: F covers G

$$(PQ)^+_F = \{P, Q, R, S\}$$

$$(R)^+_F = \{R, S\}$$

Hence F covers G, but G does not cover F
Hence F covers G, but G does not cover F as the closure of Q using the FD set G is not the same as using the FD set F.
G does not the cover F as

Q+ = Q, but in F, FD : Q → R that G does not hold.

R+ = RS

PQ+ = PQRS

Therefore, F ≠ G

Rack Your Brain

R(A, B, C, D, E) has two FD sets F and G

$$F = \begin{Bmatrix} A \rightarrow B, & D \rightarrow AC \\ AB \rightarrow C & D \rightarrow E \end{Bmatrix} \qquad G = \begin{Bmatrix} A \rightarrow BC \\ D \rightarrow AE \end{Bmatrix}$$

Check if F covers G

Finding the minimal cover:

Canonical/minimal cover is the reduced form of functional dependency also known as irreducible set.

- It is free from all extraneous FDs.
- The closure of canonical cover is same as that of the given set of functional dependencies.
- Canonical cover is not unique.
- It reduces the computation time.

Steps to find canonical cover:

Given a set of Functional dependencies, F:

1) Start with F.

2) Remove all trivial functional dependencies.

3) Repeated apply until no changes are possible.

- Union simplification
- RHS simplification
- LHS simplification

4) Result is a minimal cover.

SOLVED EXAMPLES

Q6 **FD = { E → G,**
G → S,
E → S
}

Sol: Using union rule:

$E \rightarrow GS$

$G \rightarrow S$

Simplifying RHS,

$E \rightarrow G$

$G \rightarrow S$

This is the canonical cover obtained from given Functional Dependencies.

OR

There is one more explanatory way to solve this.

I) Write the given set of FDs such that each FD on LHS has only one attribute.

II) Check each FD one by one to see if the obtained set of Functional dependencies is essential or not.

Now, there are two cases:

Case I: If the results are the same, it means that FD is non-essential and can be removed.

Case II: If the results are different, it indicates that Functional dependency is necessary, and FD should not be deleted.

III) Now, check for any FD that contains more than one attribute on its left side. If there exists such FD, then check if their LHS can be reduced with the help of the following steps:

- Compute the closure of possible subsets of LHS of that FD.
- Replace the LHS with the subset if any subset yields the same closure result as the complete left side.

Previous Years' Question

Suppose the following Functional dependencies hold on a relation U with attributes P, Q, R, S, and T:

- $P \rightarrow QR$
- $RS \rightarrow T$

Which of the following Functional dependencies can be inferred from the above Functional dependencies?

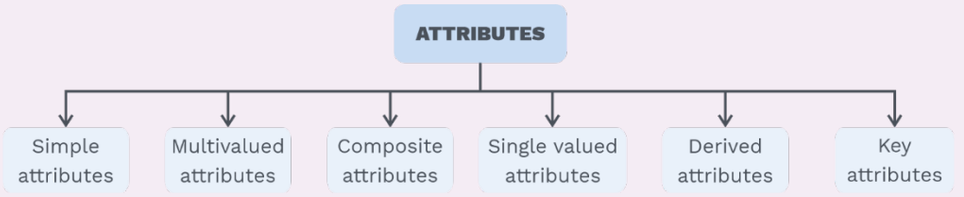
1) $PS \rightarrow T$ **2)** $R \rightarrow T$

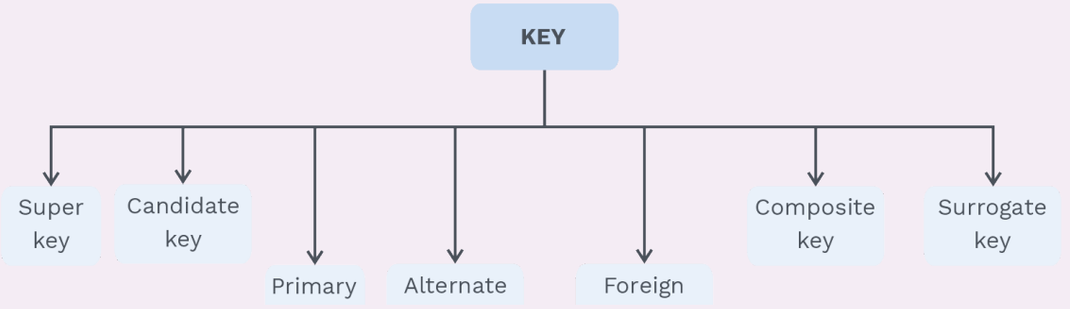
3) $P \rightarrow R$ **4)** $PS \rightarrow Q$

Sol: 1, 2, 3 **(GATE-CSE Set-2 2021)**

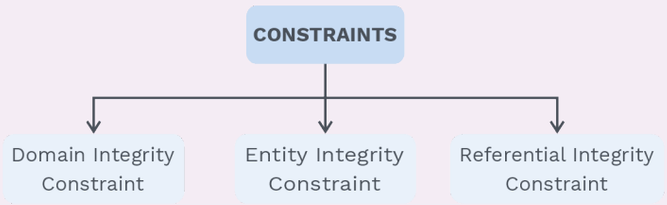
Chapter Summary

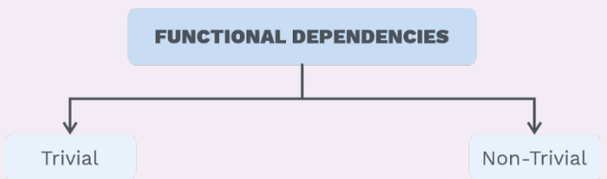
- A DBMS is a software that supports the management of a large collection of data.
- DBMS provides the user with data independence, efficient data access, automatic data integrity and security.
- Database design as six steps:
  - **a)** Requirement analysis
  - **b)** Conceptual database design
  - **c)** Logical database design
  - **d)** Schema refinement
  - **e)** Physical database design
  - **f)** Security design





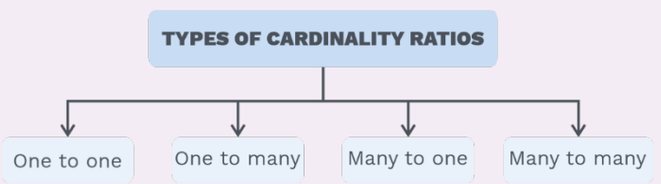
- The relational database modelling represents the database as a collection of relations.
- Properties of relation:
  - **a)** Name of relation should be distinct
  - **b)** Tuple should not have duplicate value
  - **c)** Name of every attribute should be distinct

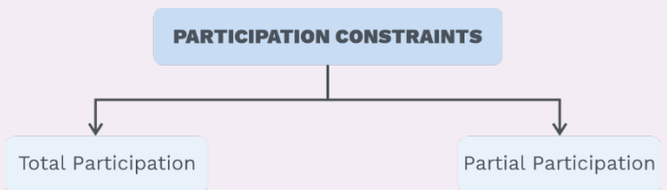




- **Armstrong's axioms:**
  - **a)** Reflexivity: If $B \subseteq A$, then $A \rightarrow B$
  - **b)** Transitivity: If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
  - **c)** Augmentation: If A → B then AC → BC where C is an attribute–sets

  From the above axioms, we can derive :
  - **d)** Decomposition: IF $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$
  - **e)** Composition: IF $A \rightarrow B$ and $C \rightarrow D$, then $AC \rightarrow BD$
  - **f)** Additive: IF $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow BC$





2 Normalization

We have discussed functional dependencies in chapter 1. In this chapter, we will dive deep and learn further things related to Functional dependencies.

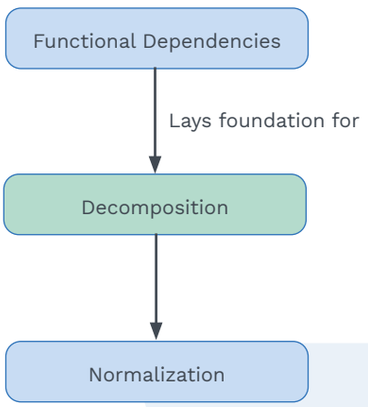


Fig. 2.1

First of all, let's revise some important points from previous chapter.

- Functional dependency is defined as a constraint between sets of attributes in a relation.
- We use '→' notation, which means LHS is functionally dependent on RHS.

We have covered following things previously,

- Functional dependencies
- Types of functional dependencies

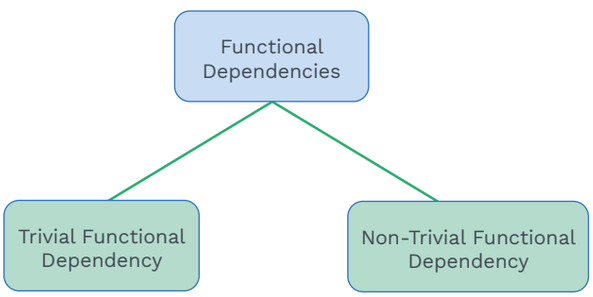


Fig. 2.2

- Rules of functional dependency
- Attribute closure
- Minimal cover
- A Problem caused by redundancy
- Equivalence set of functional dependency

Normalization

- "Normalization is used to organise data properly in tables.
- It is used to reduce the redundancy from a relation or set of relations."
- It is used to eliminate anomalies like insertion, deletion and updation.

Note:

Two or more relations when stored in single table may cause redundancy.
We can understand anomalies with the following example:

Let us consider there are three relations:

i) Sid → Sname, Sage
ii) Uid → Uname, Professor
iii) Sid, Uid → Fees

Combining these three independent relations into a single RDBMS table.

S_id	S_name	S_age	Uid	Uname	Professor	Fee
1	Arya	18	101	DU	Rosen	1000
2	Arya	18	101	DU	Rosen	1200
3	Bhumi	19	102	DU	Rosen	1200
4	Swati	17	102	MDU	Galvin	9000
4	Swati	17	103	MDU	Galvin	5000

Table 2.1

Now,

1) If we want to delete any student's data, the corresponding course data will also be deleted. This is a deletion anomaly. Similarly, if we are going to delete Uid, then corresponding Sid will also get deleted.
2) If we want to insert a university, then we have to add student information as well, which will cause an insertion anomaly.
3) If we want to update, for example, student age, this may cause inconsistency and give rise to an updation anomaly.

OR

If we want to delete/update referenced attribute value used by some referencing attribute, RDBMS won't allow deletion from referenced relation.

Normalization is process of decomposing relation into sub-relation, which must be in normal form.

Note:
If redundancy is reduced, then database anomalies can also be reduced.

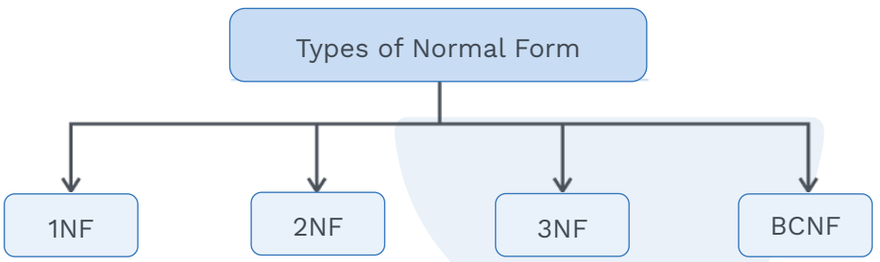


Fig. 2.3 Types of Normal Forms

2.1 FIRST NORMAL FORM (1NF)

"Any relation is said to be in 1NF if it has atomic value. Formally, an attribute of a table cannot hold multiple / Composite / Complex values."

Example: Following relation is not in 1NF because of attribute "S_course" having multiple values for a single student.

S_id	S_name	S_age	S_contact	S_cource
1	Abhi	20	9801010101	DS, Algo
2	Swati	19	9802020202	OS, CoA
3	Swarg	17	9803030303	TOC, CD
4	Shivam	21	9804040404	CN, DBMS

Table 2.2 First Normal Form

The above table can be converted into 1NF.

S_id	S_name	S_age	S_contact	S_cource
1	Abhi	20	9801010101	DS
1	Abhi	20	9801010101	Algo
2	Swati	19	9802020202	OS
2	Swati	19	9802020202	CoA
3	Swarg	17	9803030303	TOC
3	Swarg	17	9803030303	CD
4	Shivam	21	9804040404	CN
4	Shivam	21	9804040404	DBMS

Table 2.3

This is in 1NF, but the primary key will be {Sid, S_name, S_contact, S_course}

Issue with 1NF: 1NF is based on redundancy, Other normal forms came into existence, because of this problem To get rid of these anomalies, Normalization came into the picture.

Rack Your Brain

Consider the following table which is not in 1NF, how many rows will be there when converted to 1NF.

Student_id	Student_name	Contact
1	A	0101010101, 0202020202
2	B	0303030303
3	C	0404040404, 0505050505
4	D	0606060606

2.2 SECOND NORMAL FORM (2NF)

A relation will be in 2NF iff

- "It is in 1NF
- It has no Partial Dependency

Partial dependency: "When a proper subset of a candidate is functionally dependent on non-prime attribute, this type is termed as Partial Dependency

OR

$$X \rightarrow A$$

X: Proper subset of a candidate key.
A: Non-prime attribute

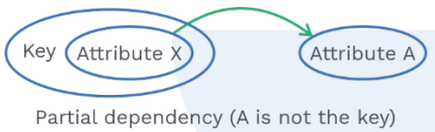


Fig. 2.4

Example: Suppose a table student is given with FD = {Sid → Sname}
Student

Sid	Sname	Course_id
S_1	n_1	C_1
S_1	n_1	C_2
S_2	n_2	C_2
S_2	n_2	C_3

Table 2.4

Here, Candidate key = {(Sid, Course_id)}
Prime attribute = {Sid, Course_id}
Non-prime attribute = {Sname}

Given FD = [Sid → Sname]

Here, Sid is proper subset of candidate key and Sname is a non-prime attribute, therefore giving Partial Dependency,

SOLVED EXAMPLES

Q2 **Given a relation R(ABCDEFG) is already in 1NF with set of functional dependencies F = {AB → C, B → F, A → C, E → G}, check if relation is in 2NF**

Sol: Here, Candidate key = {ABDE}

Therefore, Prime attribute = {A, B, D, E}

Non-prime attribute = {C, F, G}

Here, AB → C, B → F, A → C, E → G are partial dependencies

Therefore, given relation is not in 2NF.

Rack Your Brain

Consider the table student with following functional dependencies

S_id → S_name
S_id → S_address
S_id → S_contact

Determine whether the relation is in 2NF or not?

Previous Years' Question

A database of research articles in a journal uses the following schema.
(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, YEAR, PRICE)

The primary key is (VOLUME, NUMBER, STARTPAGE, ENDPAGE) and the following functional dependencies exist in the schema.

(VOLUME, NUMBER, STARTPAGE, ENDPAGE)	→	TITLE
(VOLUME, NUMBER)	→	YEAR
(VOLUME, NUMBER, STARTPAGE, ENDPAGE)	→	PRICE

The database is redesigned to use the following schemas
(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, PRICE)
(VOLUME, NUMBER, YEAR)

Previous Years' Question_continued

Which is the weakest normal form that the new database satisfies, but the old one does not?

1) 1NF **2)** 2NF
3) 3NF **4)** BCNF
Sol: 2)

2.3 THIRD NORMAL FORM (3NF)

A relation R is in 3NF if for every FD X → A that holds over R, where X is a subset of the attributes of R and A be an attribute of R, one of the following statement is true:

- X is a trivial FD, or
- X is super key, or
- A is part of some key for R.

"Formally, we can say X → a; for this to be in 3NF, either X has to be superkey or 'a' has to be a prime attribute.
Suppose that a dependency X → A causes a violation of 3NF. There are two cases.

i) X is a proper subset of some key K, i.e. partial dependency.
ii) X is not a proper subset of any key, i.e. transitive dependency because it means we have a chain of dependencies K → X → A."

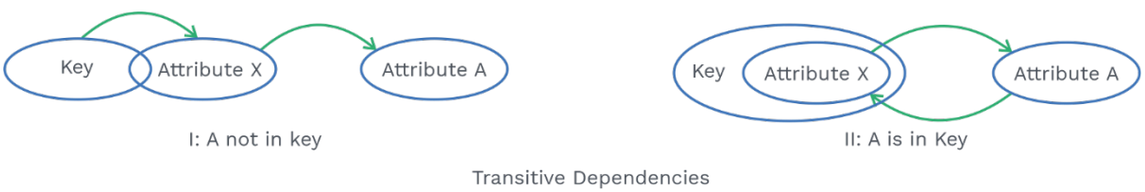


Fig.2.5

Q3 **Given a relation R (PQRSTU) with functional dependencies F = {PQ → R, R → S, S → T, Q → U}.**
Check for 3NF.

Sol: Here, Candidate key = {PQ}

Prime attribute = {P, Q}

Non-prime attribute = {R, S, T, U}

Now, Q → U is a Partial Dependency

R → S, S → T give rise to transitive dependency

Therefore, it is neither in 2NF nor in 3NF.

Rack Your Brain

Consider the table student with the following functional dependencies

S_id → S_name
S_id → S_address
S_id → S_contact

Determine whether the relation is in 3NF or not?

2.4 BOYCE-CODD NORMAL FORM (BCNF)

BCNF stands for Boyce-Codd Normal Form. Given a relation R and functional dependency, X → Y is in BCNF iff.

- X ⊇ Y, trivial
- X is a superkey

Note:

Every BCNF is in 3NF. There are very few examples which are in 3NF but not in BCNF.

If the relation is not in 1 NF, then it does not satisfy RDBMS rules.

Q4 **Given a relation R (ABCD) with set of functional dependencies F = {AB → CD, D → A}. Check for BCNF.**

Sol: Here, Candidate key = {AB, BD}

Prime attribute = {A, B, D}

Non-prime attribute = {C}

In FD, D → A, D is not a superkey.

Therefore, it is not in BCNF.

Rack Your Brain

Consider the table student with the following functional dependencies

S_id → S_name
S_id → S_address
S_id → S_contact

Determine whether the relation is in BCNF or not?

Rack Your Brain

Consider the following statements:
S1: Every relation which is in BCNF is also in 3NF.
S2: Every relation which is in 4NF is also in BCNF.

Which of the following option is True?

1) Only S1 is true
2) Only S2 is true
3) None of them is true
4) Both of them are true

Rack Your Brain

Given a relation R (PQRSTUVWXY) and FD set F = {PQ → R, P → ST, Q → T, T → VW, S → XY}.
Check for 2NF, 3NF and BCNF.

BCNF and multivalued dependencies:

- Where redundancy occurs because of presence of multivalued attributes, then this comes under 4NF.
- Multivalued Dependency is represented by either using '→ →' or ' →→ '.

Multivalued dependency:

When two attributes are independent of each other but on a third attribute, this gives rise to multi valued dependency.

Properties of multivalued dependency:

i) If $A \rightarrow\rightarrow B$ then $A \rightarrow\rightarrow D$ when D = R [Complement property]

ii) If $A \supseteq B$ then $A \rightarrow\rightarrow B$

$A \cup B$ then $A \rightarrow\rightarrow B$

iii) Augmentation property:
$A \rightarrow\rightarrow B$ and $C \supseteq D$ (This is trivial dependency, $C \rightarrow D$ iff $D \subseteq E$) but augmentation property holds true for the non-trivial property as well, $AC \rightarrow\rightarrow BD$

iv) Transitivity property:
If $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow C$ then $A \rightarrow\rightarrow C$

v) If $A \rightarrow B$, then $A \rightarrow\rightarrow B$

Note:

- If $A \rightarrow\rightarrow BC$ then MVD does not follow $A \rightarrow\rightarrow B$ and $A \rightarrow\rightarrow C$.
- If $A \rightarrow\rightarrow B$ and $A \rightarrow\rightarrow C$, MVD does not follow $A \rightarrow\rightarrow BC$.

Fourth normal form(4NF)

It is an extension of BCNF. A relation is in 4NF iff

i) It is in BCNF

ii) For MVD $A \rightarrow\rightarrow B$. A should be super key, and it should be determined using functional dependency, not using MVD.

Example: Consider the following table T,

T

A	B	C
1	P	R
1	P	S
1	Q	R
1	Q	S
2	Q	S

Let T is decomposed into T_1 and T_2

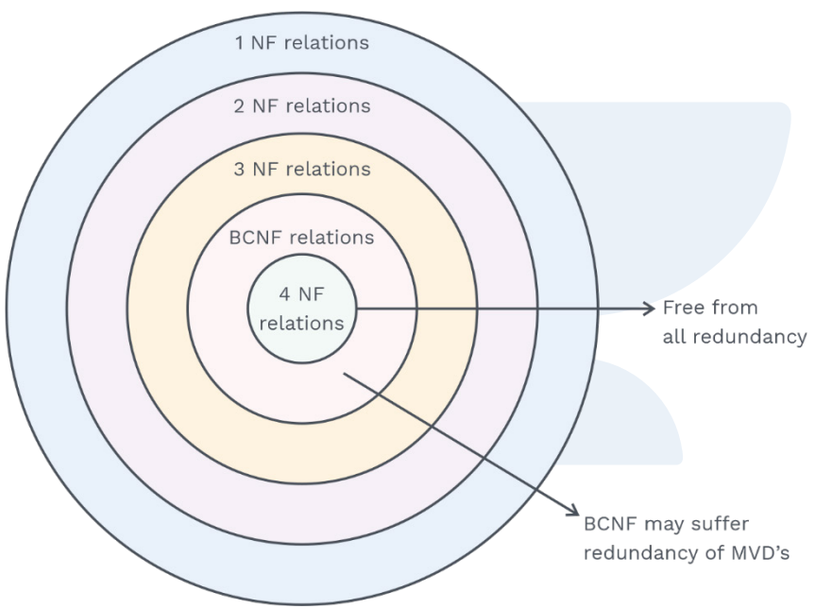
T_1

A	B
1	P
1	Q
2	Q

T_2

A	B
1	R
1	S
2	S

Here, Multivalued Dependency = {A $\rightarrow\!\!\rightarrow$ B, A $\rightarrow\!\!\rightarrow$ C}



Previous Years' Question

Given the following two statements:

S1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.

S2: AB $\rightarrow$ C, D $\rightarrow$ E, E $\rightarrow$ C is a minimal cover for the set of functional dependenciesAB $\rightarrow$ C, D $\rightarrow$ E, AB $\rightarrow$ E, E $\rightarrow$ C.

Which one of the following is correct?

1) S1 is TRUE, and S2 is FALSE. **2)** Both S1 and S2 are TRUE.

3) S1 is FALSE, and S2 is TRUE. **4)** Both S1 and S2 are FALSE.

Sol: 1)

Previous Years' Question

Let the set of functional dependencies F = {QR → S, R → P, S → Q} hold on a relation schema X = (PQRS). X is not in BCNF. Suppose X is decomposed into two schemas, Y and Z, where Y = (PR) and Z = (QRS).

Consider the two statements given below.

I) Both Y and Z are in BCNF

II) Decomposition of X into Y and Z is dependency preserving and lossless

Which of the above statements is/are correct?

1) Both I and II **2)** I only

3) II only **4)** Neither I nor II

Sol: 3)

2.5 PROPERTIES OF DECOMPOSITION

When some relation is not in normal form, then that relation is decomposed into multiple relations, which is used to eliminate anomalies, redundancy and inconsistencies.

The two properties of decomposition are :

- Lossless Decomposition
- Dependency Preserving decomposition

Lossless join decomposition

Decomposition without any loss of data is termed lossless join decomposition.

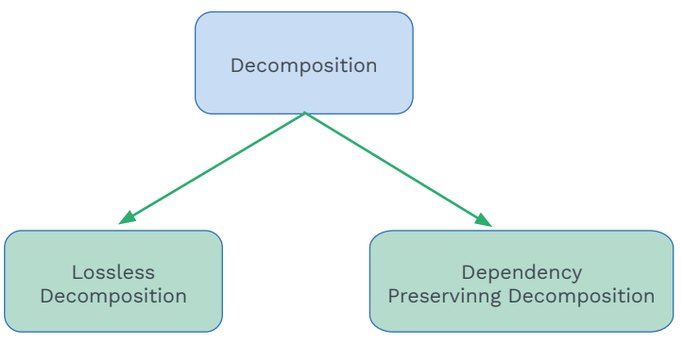
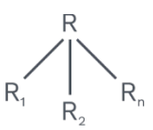


Fig. 2.5

Example: Generalization,



$[R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n] \in R$

- If $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) = R$, then it is lossless join decomposition.
- If $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \supset R$, then, it is lossy decomposition.

Note:
In lossy decomposition, some extra tuples are generated, called spurious tuples.

Example: Let a relation R with attributes emp_id, emp_name, dept_id.

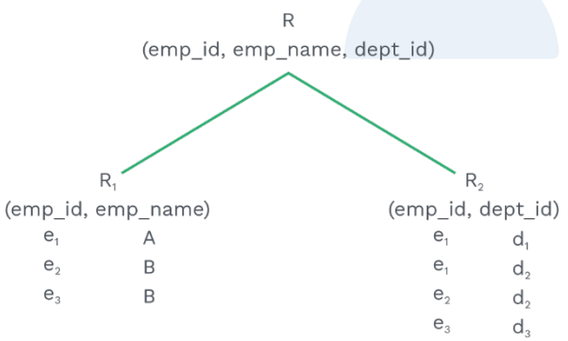
R

Emp_id	Emp_name	Dept_id
e_1	A	d_1
e_1	A	d_2
e_2	B	d_2
e_3	B	d_3

Table 2.5

- The candidate key of the above relation will be emp_id, dept_id

Case I: Let this relation is decomposed into two relations



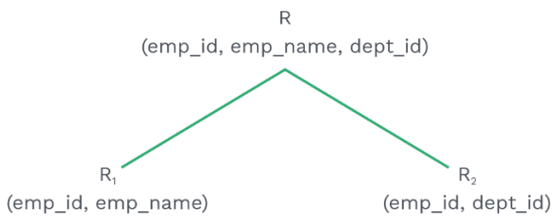
- Now joining R_1 and R_2, we get;

Emp_id	Emp_name	Dept_id
e_1	A	d_1
e_1	A	d_2
e_2	B	d_2
e_3	B	d_3

Table 2.6

$R_1 \bowtie R_2$ = R. that is lossless decomposition

Case II: Let the given relation R is decomposed into,



R_1

Emp_id	Emp_name
e_1	A
e_1	A
e_2	B
e_3	B

R_2

Emp_id	Dept_id
e_1	d_1
e_1	d_2
e_2	d_2
e_3	d_3

Now, $R_1 \bowtie R_2$ =

Emp_id	Emp_name	Dept_id
e_1	A	d_1
e_1	A	d_2
e_2	B	d_2
e_2	B	d_3
e_3	B	d_2
e_3	B	d_3

Table 2.7

Now, as we can see $(R_1 \bowtie R_2) \supset R$, therefore lossy join decomposition.

SOLVED EXAMPLES

Q1 **Given a relation R(A, B, C) with functional dependency {A → B, B → C}. Check if given relation is lossy or lossless, if relation is decomposed into**

i) R_1(AB) and R_2(BC) **ii) R_1(AC) and R_2(BC)**

Sol: Let's try to solve this question by taking counter example,

Now,

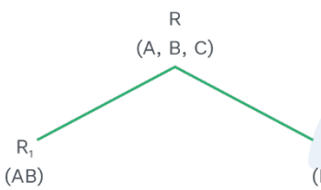
R

A	B	C
1	2	1
3	3	1
4	5	3
5	5	3
9	4	3

Candidate key = A

Now,

i) R_1(AB) and R_2(BC)



R_1

A	B
1	2
3	3
4	5
5	5
9	4

R_2

B	C
2	1
3	1
5	3
5	3
4	3

$\Rightarrow$ Taking the join of R_1 and R_2, we get R. Therefore, lossless join decomposition.

ii) R_1(AC) and R_2(BC)

As C is the common attribute and is not unique. Therefore, decomposition is lossy, $(R_1 \bowtie R_2) \supset R$.

Note:

Any relation is lossless iff

- $(R1 \cup R_2) = R$
- $(R1 \cap R_2) = R_1$ or $(R1 \cap R_2) = R_2$, formally, either $R1 \cap R_2$ is superkey/ candidate key of R_1 or $R1 \cap R_2$ is superkey of R_2.

Q2 **Consider a relation R(ABCDE) with functional dependencies, {AB → C, C → D, B → E} test whether given decompositions are lossless or lossy. Consider a relation R(ABCDE) with functional dependencies, {AB → C, C → D, B → E}**

Sol:

i) R1(ABC) and R2(CD)

$\Rightarrow$ $R_1 \cup R_2 \neq R$, $R_1 \cup R_2$ is giving ABCD, E being lost.

Therefore, lossy.

ii) R_1(ABC) and R_2(DE)

$\Rightarrow$ As, $R_1 \cup R_2 = R$, but no common attribute $(R_1 \cap R_2 = \phi)$

Therefore, lossy.

iii) R_1(ABC) and R_2(CDE)

$\Rightarrow$ As, $R_1 \cup R_2 = R$

$R_1 \cap R_2 = C$

But C is not a Candidate key, neither of R_1, nor R_2.
Therefore, lossy.

iv) R_1(ABCD) and R_2(BE)

$\Rightarrow$ $R_1 \cup R_2 = R$

$R_1 \cap R_2 = B$, which is the candidate key of R_2.

Therefore, lossless join decomposition.

v) R_1(ABCD) and R_2(ABE)

$\Rightarrow$ $R_1 \cup R_2 = R$

$R_1 \cap R_2 = \{A, B\}$

Now, $(AB)^+$ = ABCDE, superkey in both the relations
Therefore, lossless join.

Rack Your Brain

Consider the following relation R (A, B, C, D, E)

A	B	C	D	E
1	101	A	Alia	CSE
2	102	B	Swati	EE
1	103	A	Shruti	ME
3	104	C	Swayam	CE
4	105	D	Shivam	IT

If the given relation is decomposed into two relations, R_1 (ABC) and R_2 (CDE), then check if they form lossless/lossy decomposition.

Dependency preserving decomposition:

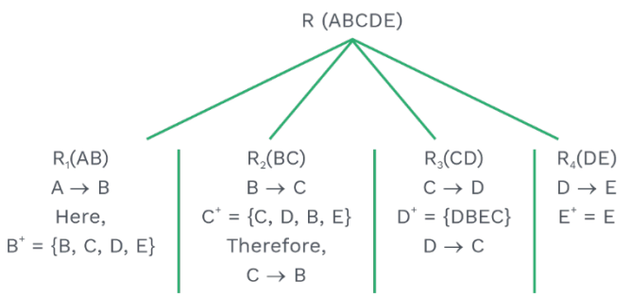
A relation R is decomposed into a number of relations R_1, R_2, ... R_n with F_1, F_2, ... F_n Functional dependencies.

Then, $\{F_1 \cup F_2 \cup F_3 \cup \quad F_n\} \subseteq F$

- If $\{F_1 \cup F_2 \cup F_3 \quad \cup F_n\} = F$, then decomposition is dependency preserving decomposition, i.e. every FD of F is present in sub relation.
- If $\{F_1 \cup F_2 \cup F_3 \quad \cup F_n\} \subset F$, then decomposition is not dependency preserving decomposition.

Example: Given a relation R(ABCDE) with functional dependencies, {A → B, B → C,C → D, D → BE} decomposed to R_1(AB), R_2(BC), R_3(CD), R_4(DE). Check whether Dependency is preserved or not?

Sol:



Now, $A \rightarrow B$
$B \rightarrow C$
$C \rightarrow D$
$D \rightarrow BE$, because $D^+ = \{DBEC\}$

Therefore, $F_1 \cup F_2 \cup F_3 \cup F_4 = F$

Hence, Dependency is preserved.

Rack Your Brain

Given a relation R(ABCDEF) with functional dependencies {A → BCDE, BC → ADF, B → F, D → E} is decomposed into ABC, BCD, BF, DE. Check if dependency can be preserved or not.

Rack Your Brain

Consider a table Employee:

Emp_id	Emp_name	Salary
1	A	50,000
2	B	55,000
3	C	60,000

This relation is decomposed into the relations emp_name and salary.

emp_name

Emp_id	Emp_name
A	1
B	2
C	3

salary

Emp_id	Salary
1	50,000
2	55,000
3	60,000

Check whether it satisfies Dependency preserving property or not?

Previous Years' Question

Let R(A, B, C, D) be a relational schema with the following functional dependencies:
A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D and D $\rightarrow$ B. The decomposition of R into (A, B), (B, C), (B, D)

1) gives a lossless join and is dependency preserving
2) gives a lossless join but is not dependency preserving
3) does not give a lossless join but is dependency preserving
4) does not give a lossless join and is not dependency preserving

Sol: 3)

Previous Years' Question

Suppose the following functional dependencies hold on a relation U with attributes P, Q, R, S, and T:

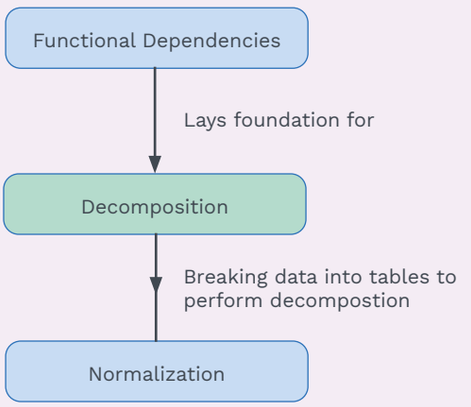
- P $\rightarrow$ QR
- RS $\rightarrow$ TS

Which of the following functional dependencies can be inferred from the above functional dependencies?

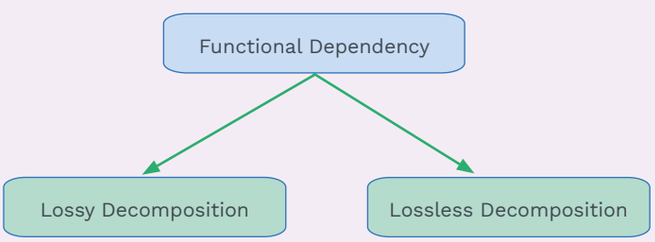
1) PS $\rightarrow$ T
2) R $\rightarrow$ T
3) P $\rightarrow$ R
4) PS $\rightarrow$ Q

Sol: 1)

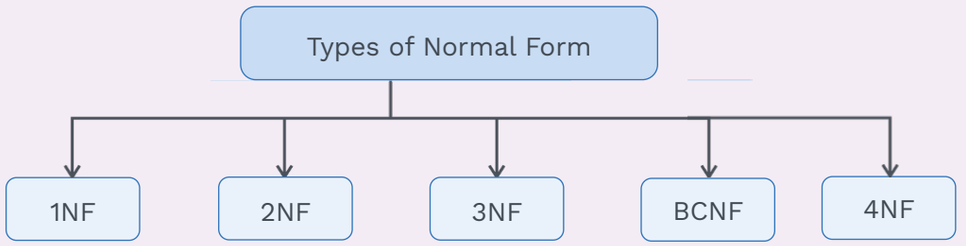
Chapter Summary

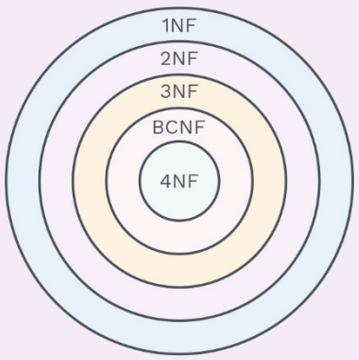
- 

- "Properties of decomposition:
  i) Lossless decomposition.
  ii) Dependency preserving decomposition."

- 

- Lossless decomposition:
  i) If $(R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) = R$ then it is lossless Join decomposition
  ii) if $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n) \supset R$ then it is lossy.
- Dependency preserving decomposition
  i) If $\{F_1 \cup F_2 \cup F_3 \ldots \cup F_n\} = F$, then decomposition is dependency preserving property.
  ii) If $\{F_1 \cup F_2 \cup F_3 \ldots \cup F_n\} \subset F$, then it is not dependency preserving.


Types of Normal Form
1NF
2NF
3NF
BCNF
4NF


1NF
2NF
3NF
BCNF
4NF

3 SQL

3.1 SQL

- SQL stands for Structured Query Language.
- We use SQL as commercial relational database language.
- Initially, SQL was known as Structured English QUERY Language(SEQUEL). It was designed and implemented by IBM Research.
- SQL is a standard way to add, delete, modify and obtain the data.
- SQL is a declarative programming language. It means we need to declare what we want, but we need not focus on how to get data. We do not specify step by step procedure to obtain data.

Features of SQL languages are:

Embedded SQL:
Embedded SQL is a feature using which a host language can call an SQL code.

Dynamic SQL:
In dynamic SQL, we perform queries at run time.

Security: SQL provides mechanisms to control user's access to data objects such as tables and views.

Transaction management:
A user is allowed to explicitly control how a transaction to be executed using various commands.

Execution of SQL query:
The DBMS performs a number of steps while executing a query. The conceptual view of this process is shown below.

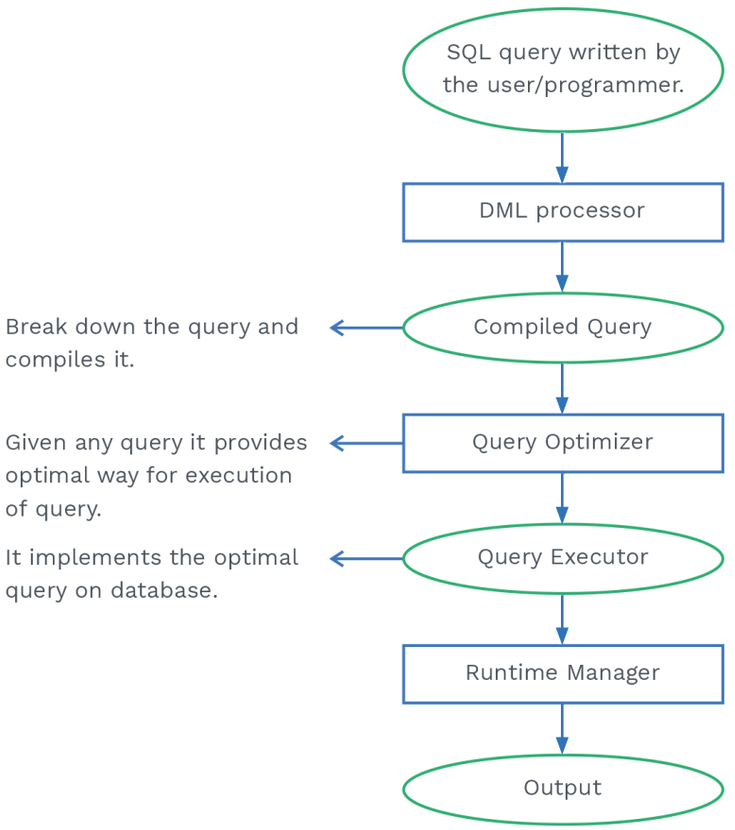


Fig. 3.1 Execution of SQL Query

- Terms like table, row and column in SQL are used for the relational model terms like relation, tuple and attribute, respectively. We can use the corresponding terms interchangeably.
- SQL uses certain commands. These SQL commands are mainly:
  1) DDL: Data Definition Language
  2) DML: Data Manipulation Language
  3) DQL: Data Query Language
  4) DCL: Data Control Language

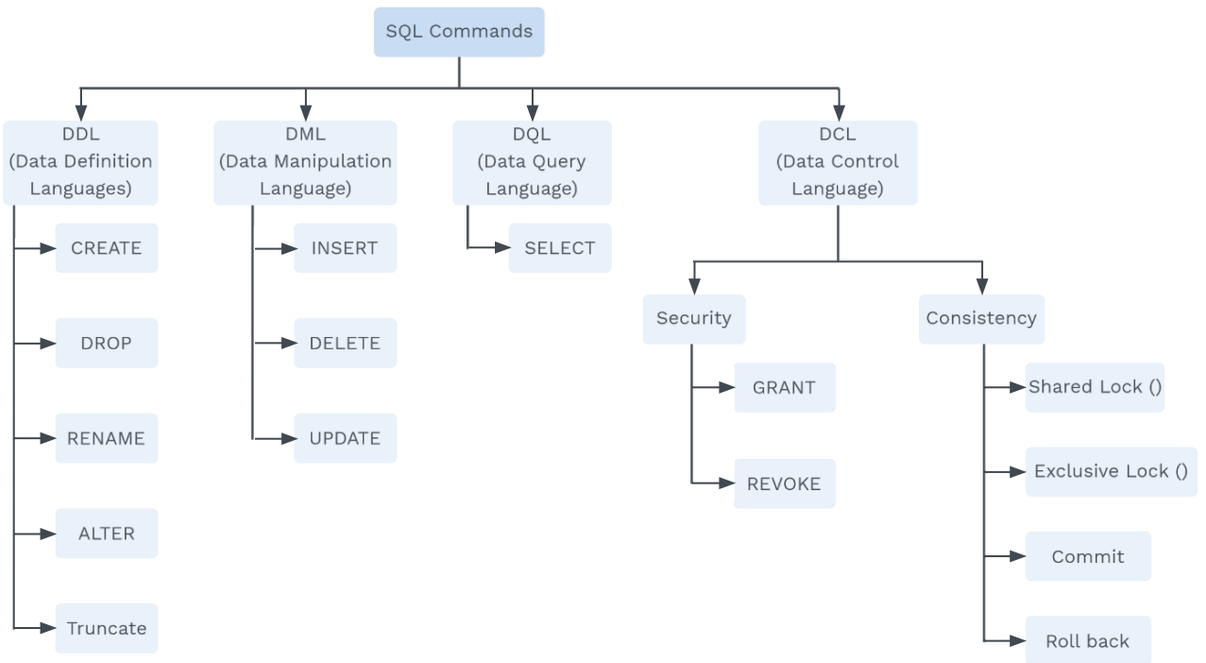


Fig. 3.2 SQL Commands

DDL (Data Definition Language):

- DDL is used for creating, deleting and modifying tables.
- Common examples of DDL statements include CREATE, ALTER and DROP.

Example: DROP TABLE Employees

DML (Data Manipulation Language):

- DML is used for inserting, deleting and modifying data in database.

Example: INSERT INTO Employees (Fname, Lname) VALUES ('Shikha', 'Sharma')

DQL (Data Query Language):

- Data query language performs queries on the data within schema objects.
- The DQL commands are used to get the relation based on whatever query we pass to it.
- To retrieve data from Database, we use SELECT Command.

DCL (Data Control Language):

- Data Control Language feature is used to control access to data stored in a database.
- GRANT and REVOKE are examples of DCL.
- GRANT: It is used to allow specified users to perform specific tasks.
- REVOKE: It is used to remove the user accessibility to database objects.

Basic structure of SQL queries:

- The basic structure of an SQL queries:

SELECT, **FROM** and **WHERE:**

- The SELECT clause projects the output desired attributes. It bears similarity with the projection operation of relational algebra.
- The FROM clause bears resemblance with the cartesian product of the tables involved.
- The WHERE clause deals with the specification of the condition to be followed while fetching a record from the resultant of FROM clause. The work which is done by WHERE clause is the same as the work done by selection operation of the relational algebra.
- A typical SQL query has the form:
  SELECT <attributes list>
  FROM <table list>
  WHERE <condition>;
  The query is equivalent to the relational-algebra expression:

  $$(|A_1| \cup |A_2|) = |A_1| + |A_2| - |A_1| \cap |A_2| = 2^{n-1} + 2^{n-1} - 2^{n-2} = 2^n - 2^{n-2}$$

  Each A_i is an attribute, and each r_i is a relation and c is a condition. The only difference is the result of a relational algebra expression do not produce duplicate rows, but the result of the SQL query might produce more than one copy of some rows.
- SQL first does the cartesian product of the tables present in the FROM clause, then it performs a relational-algebra selection using the condition specified in the WHERE clause and then project the result onto the attributes of the SELECT clause.

The SELECT clause:

Syntax: SELECT < attribute-list > FROM < table-list>;
Let us realise the above clauses through this query:
“Find the names of all employees in the Employee relation”.

Sol: SELECT Emp-name FROM Employee;

- This query will result a table that consists of a single attribute with the Empname as a heading.
- SQL permits duplicates in the results.
- Thus, the above query will give each employee name once for every row in which it appears in the Employee table.
- There are cases where we want to eliminate duplicates; for that, we use keyword “distinct” after “select”.

Example: SELECT DISTINCT Emp-name FROM Employee;

- SELECT clause might also contain arithmetic expressions that involve the operators +, –, /, and * operating on constants or attributes of tuples.

Example: SELECT Emp-name, Salary * 200 FROM Employee;
This will return a table containing columns Emp-name and Salary with the column Salary is multiplied by 200.

The WHERE clause:
Let us consider the query on relation Employee:
"Find the names of all employees who works for department CS".
In SQL: SELECT Emp-name FROM Employee WHERE dept = CS;

The FROM clause:
The FROM clause by itself defines a cartesian product of the tables in the clause.
Let us consider the query in relation to Employee and Department:
"Retrieve the employee id and name of the employees who works for the IT department".

Department

Dept_id	Dept_name
1	CSE
2	IT
3	ECE
4	ME

Employee

Emp_id	Dept_no	Emp_name
1	3	Raju
2	4	Rima
3	2	Puja
4	4	Nisha
5	1	Koyal

Table 3.1

In SQL: SELECT Emp_id, Emp_name FROM Employee, Department WHERE Dept_no = Dept_id and Dept_name = 'IT';

- The above SQL query will give the result as {3, puja}
  Consider the following table definitions:
  Shopkeepers (Sid: integer, Shname: string, Rating: integer, Age: integer)

Sid	Shname	Rating	Age
1	Ramu	7	20
2	Rupa	2	29
4	Kajal	4	22
5	Koyal	5	23
10	Mahima	8	25
9	Gopal	9	24
7	Rupa	3	29

Table 3.2

PRACTICE QUESTIONS

Q1 **Find the names and ages of all Shopkeepers.**

Sol: SELECT Shname, Age FROM Shopkeepers;

Shname	Age
Ramu	20
Rupa	29
Kajal	22
Koyal	23
Mahima	25
Gopal	24
Rupa	29

Q2 **Find the names and ages of all Shopkeepers having no duplicate names and ages.**

Sol: SELECT DISTINCT Shname, Age from Shopkeepers;

Output:

Shname	Age
Ramu	20
Rupa	29
Kajal	22
Koyal	23
Mahima	25
Gopal	24

We will all distinct <Shname, Age> if two or more Shopkeepers have the same name and age, the answer will contain only one pair.

Q3 **Find all the Shopkeepers with a rating above 5.**

Sol: SELECT * FROM Shopkeepers WHERE rating > 5;

Output:

Sid	Shname	Rating	Age
1	Ramu	7	20
10	Mahima	8	25
9	Gopal	9	24

Note:

When we want to retrieve all columns, SQL provides a convenient shorthand: SELECT *. This notation is useful for interactive querying but is poor for queries that are meant to be reused and maintained.

Note:

Hey Learners!!!
Now, we will discuss DISTINCT keyword.
By default, an SQL query contains duplicates in the result. In order to get the distinct result, we use DISTINCT Keyword.
We use DISTINCT to eliminate duplicate tuples.

The RENAME operation:

- SQL provides a mechanism for renaming both relations and attributes.
- It uses the AS clause.

Example: Consider the following given relation Employee and for each employee retrieve employee's id, name, salary and the name of his/her immediate supervisor.

Employee

Emp_id	Dept_id	Sup_id	Sal	Emp_name
1	1	7	1100	Sindhu
2	3	4	2200	Somya
3	1	7	2100	Santosh
4	4	7	3000	Megha
5	2	4	2000	Murali
6	1	7	5000	Ravi
7	1	3	1800	Rupa

Table 3.3

Sol: SELECT E. Emp_id AS "Employee_id", E. sal AS "Salary", E. Emp_name AS "Employee_name", S. Emp_name AS "Supervisor_name" FROM Employee AS E, Employee AS S WHERE E. Sup_id = S. Emp_id;

Rack Your Brain

Consider the following relation Department and Employee. Compute an SQL query that lists out employees names and department names.

Department

Dept_id	Dept_name
1	Accouting
2	Sales
3	Research
4	Aviation

Employee

Emp_id	Dept_id	Salary	Emp_name
1	1	1100	Koyal
2	3	2200	Komal
3	4	2100	Raj
4	2	3000	Sonu

String operation:

- Pattern matching is the most frequently used operation on strings that uses operator named as LIKE.
- There are two special characters that is used to describe patterns:
  1) % (percent): The % character matches any substring.
  2) _ (underscore): The underscore (_) character matches any character.
- 'A%' matches any string that starts (begins) with 'A'.
- '%ai%' matches any string containing "ai" as a substring. For example, 'Rain', 'Paid', 'Sail' etc.
- '_ _' matches any string of exactly two characters.
- '_ _ _ %' matches any string of at least three characters.
- SQL uses the LIKE comparison operator to express patterns.

Note:
Patterns are case sensitive, i.e. uppercase characters are different than lowercase characters or vice-versa.

Example: Following is the given relation Student.

Name	Marks	Rank	Email
Sindhu	78	188	abc@gmail.com
Somya	92	15	def@gmail.com
Komal	48	1500	mno@gmail.com

Table 3.4

PRACTICE QUESTIONS

Q1 **Retrieve the name of all the students whose name starts with 'S'.**

Sol: SELECT Name FROM Student WHERE Name LIKE 'S%';

Name
Sindhu
Somya

Q2 **Display the name of the students who secure 2 digit rank.**

Sol: SELECT Name FROM Student WHERE Rank LIKE '_ _';

Name
Somya

Q3 **Retrieve name, marks of all the students whose name includes substring 'ind'.**

Sol: SELECT Name, Marks FROM Student WHERE Name LIKE '%ind%';
This will result a relation containing attribute (Name, Marks) as:

Name	Marks
Sindhu	78

Note:

- SQL uses escape character to include the special characters (% and _).
- When the escape character is used just before a special pattern character, then the special pattern character will be treated as a normal character.

Example: SELECT Name FROM Student WHERE Marks LIKE '90 \ %';
gives names of the student whose marks is 90%.

Rack Your Brain

Consider the following relation Sailor.

Shopkeepers

Sid	Sname	Rating	Age
22	AMAYRA	7	20
24	ROHIT	8	24
27	ARUN	9	25
29	ASHA	5	27

Retrieve the list of ages of the shopkeepers whose name contains 'A' at start and end and is made up of three or more characters.

1) SELECT Age FROM Sailor WHERE Sname LIKE 'A _ _%';
2) SELECT Age FROM Sailor WHERE Sname LIKE 'A_%A';
3) SELECT Age FROM Sailor WHERE Sname LIKE 'A%';
4) None of these

ORDER BY:

The ORDER BY clause is used to sort/order the result of an SQL query in ascending (or) descending order.

Syntax: SELECT <column_list> FROM <table_list> ORDER BY <column_1> ASC/DESC, <column_2> ASC/DESC ...;

Note:

- By default, the ORDER BY sorts the result of an SQL query in ascending order.
- To specify the sort order, we need to specify explicitly
  1) DESC for descending order
  2) ASC for ascending order

Example: Consider the following given table R.

R

A	B	C
1	2	3
1	2	1
2	1	3
2	1	1
3	5	4
3	4	3

Table 3.5

1) SELECT A, B, C FROM R ORDER BY A, B, C;
Output:

A	B	C
1	2	1
1	2	3
2	1	1
2	1	3
3	4	3
3	5	4

Table 3.6

Here, by default A, B, C are all in ascending order.

2) SELECT A, B, C FROM R ORDER BY A DESC, B, C;
Output:

A	B	C
3	4	3
3	5	4
2	1	1
2	1	3
1	2	1
1	2	3

Table 3.7

Here, A will be ordered in descending order, and by default B, C will be ordered in ascending order.

3) SELECT A, B, C FROM R ORDER BY A DESC, B, C DESC;
Output:

A	B	C
3	4	3
3	5	4
2	1	3
2	1	1
1	2	3
1	2	1

Table 3.8

Here, A will be arranged (sorted) in descending order; by default B will be sorted in ascending order, and C will be sorted in descending order.

Set operations and Null values:

Set operations:

- The SQL operations UNION, INTERSECT and EXCEPT operate on relations.
- Similar to UNION, INTERSECTION and SET DIFFERENCE in relational algebra, the relations participating in the operations must have the same set of attributes.

Hey Learners!!!
Do you know about IN, EXISTS, ALL, ANY set operations in SQL?
Let's discuss these set operations now:

1) IN: It checks whether an element is present in a given set or not.
2) ALL, ANY: To match or correlate a specified value with the elements of a set associating these operations with a comparison operator.
3) EXISTS: To examine if a set follows an empty condition.

Note:
"EXISTS and IN can be prefixed by NOT."

Note:
- By default, the UNION, INTERSECT, and EXCEPT commands remove all the duplicates.
- If we want to retain all duplicates, we have to write UNION ALL, INTERSECT ALL, EXCEPT ALL in place of UNION, INTERSECT and EXCEPT respectively.

Example: Consider the following relation ENROLL.

ENROLL

Stud_id	Course_name	Grade
1	Maths	A
2	Physics	A
3	Chemistry	C
2	Maths	B
2	Biology	B
1	Chemistry	A
3	Physics	D
2	Chemistry	A
2	History	A

Table 3.9

PRACTICE QUESTIONS

Q1 **Retrieve the list of ids of students who got either grade 'A' or grade 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' UNION SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
1
2

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' UNION ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
1
2
2
2
1
2
2

Q2 **Retrieve the list of the ids of students who got both grade 'A' and grades 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' INTERSECT SELECT Stud_id FROM ENROLL WHERE Grade = 'B';

Output:

Stud_id
2

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' INTERSECT ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
2
2

Q3 **Retrieve the list of studs Id's who got grade 'A' but not grade 'B'.**

Sol: SELECT Stud_id FROM ENROLL WHERE Grade = 'A' EXCEPT SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
1

SELECT Stud_id FROM ENROLL WHERE Grade = 'A' EXCEPT ALL SELECT Stud_id FROM ENROLL WHERE Grade = 'B';
Output:

Stud_id
1
1
2

Previous Years' Question

SELECT Operation in SQL is equivalent to

1) The selection operation in relational algebra.
2) The selection operation in relational algebra, except that SELECT in SQL retains duplicates.
3) The projection operation in relational algebra.
4) The projection operation in relational algebra, except that SELECT in SQL retains duplicates.

Sol: Option 4) **(GATE-2021 Set-1)**

Arithmetic operators:

- SQL allows the use of arithmetic operators (+,-,*, /) in queries on attributes having numeric domains.

Example: Consider the relation schema Employee (Eid, Fname, Sex, Salary). Increase the salary of the employee by 10 percent and display the salary and employee's first name.

Sol: In SQL: SELECT Fname, Salary * 1.1 FROM Employee;

Concatenate operator:

- We use || (concatenate operator) to append two strings.
  Example: SELECT Fname || Lname as "FULL-NAME" FROM Employee;
- This will result in a relation having the attribute FULL-NAME, which contains the concatenation of Fname and Lname (i.e. first name and last name).

Employee

Fname	Lname
Komal	Jain

→

FULL-NAME
KomalJain

Between operator:

- It is a comparison operator on the Numeric domain.
- **Syntax:** SELECT column_name(s) FROM Table-Name WHERE column-name BETWEEN value-1 AND value-2;

Example: List all staff whose payscale is between 50,000 and 70,000. Relation Schema: Staff (Sid, Sname, Sex, Payscale)

Sol: SELECT * FROM Staff WHERE Payscale BETWEEN 50000 AND 70000;

- It is also a comparison operator on text and dates.

Example: SELECT * FROM Staff WHERE Hire-date BETWEEN '01-JAN-2010' AND '31-DEC-2010';

NULL values:

- In SQL, NULL value is used to indicate that the information about the value of an attribute is absent.
- The special keyword NULL is used in a predicate to check for null values.
- The output of an arithmetic expression that involves +,-,* or / is NULL if any of the input is NULL.
- The result is considered to be UNKNOWN (i.e. it may be true or false) if a comparison operation implies NULL.

Example: (1 < NULL) = UNKNOWN.

- As WHERE clause condition can involve Boolean operations (AND, OR and NOT) on the results of comparisons. Therefore, we can extend the definition of boolean operations to deal with the UNKNOWN value.

1) AND:

A	B	A and B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	UNKNOWN
FALSE	UNKNOWN	FALSE
UNKNOWN	UNKNOWN	UNKNOWN

Table 3.10

2) OR:

A	B	A or B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	TRUE
FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN

Table 3.11

2) NOT: The result of NOT UNKNOWN is UNKONWN.
NOT (TRUE) = FALSE
NOT (FALSE) = TRUE
NOT (UNKONWN) = UNKNOWN

Note:
In SQL, there are operators that can check whether the value of an attribute is NULL or not.
The IS or IS NOT clause is used to analyse if a given value is NULL or NOT.
Reason: Each NULL value is treated distinctly in SQL terminology. So, using = or <> is not at all appropriate.

Example: Consider the following relation R(A, B)

R

A	B
a	1
b	2
c	NULL
d	NULL

Table 3.12

What is the output produced by the following query?
Example: SELECT A FROM R WHERE B is NULL;
Sol: Output:

A
c
d

Table 3.13

Example: SELECT A FROM R WHERE B IS NOT NULL;
Sol: Output:

A
a
b

Table 3.14

Example: SELECT B FROM R;
Sol: Output:

B
1
2
NULL
NULL

Table 3.15

Example: SELECT DISTINCT B FROM R;
Sol: Output:

B
1
2
NULL

Table 3.16

Example: Consider the following relation Employee (Eid, Fname, Ssn, Sex, Super-Ssn)

Eid	Fname	Ssn	Sex	Super-Ssn
1	John	12	M	25
2	Ahmad	34	M	NULL
3	Ruchi	25	F	NULL

Table 3.17

Example: write a SQL query that lists the names of all employees who is not having supervisors.
Sol: SELECT Fname FROM Employee WHERE Super-Ssn IS NULL;
Output:

Fname
Ahmad
Ruchi

Table 3.18

Aggregate functions:

Definition

"Aggregate functions take a collection (a set of multiset) of values as input and return a single value."

- SQL offers 5 built-in aggregate functions:
  1) Average: AVG **2)** Minimum: MIN
  3) Maximum: MAX **4)** Total: SUM
  5) Count: COUNT

Example: Given Relation: Shopkeepers (sid, sname, rating, age)

Sid	Sname	Rating	age
1	Rupa	9	12
2	Puja	8	14
3	Sonu	10	18
4	Dilip	9	22

Table 3.19

1) Find the average age of Shopkeepers with a rating of 9.

Sol: SELECT AVG (age) Shopkeepers WHERE rating = 9;
Output: 17
2) List the name and age of the shopkeepers who are the older ones.
Sol: SELECT sname, MAX(age) FROM Shopkeepers; // Illegal query in SQL.

Note:

- In the SELECT clause, if an aggregate operation is used, then we must use aggregate operations only unless GROUP BY is present in that query.
- Thus, we cannot use MAX(age) as well as sname in the SELECT clause.
- We can use the Nested query to compute the desired answer to this question.

Nested query:

SELECT sname, age FROM Shopkeepers WHERE age = (SELECT MAX (age) from Shopkeepers);

- We will discuss later how the nested query works.

3) Count the number of Shopkeepers.

Sol: SELECT COUNT (*) FROM Shopkeepers;

⇒ COUNT (*) will consider all columns and count the number of rows.

⇒ It includes duplicates, i.e. it will not give distinct rows.

⇒ Output = 4

4) Find the sum of rating of Shopkeepers.

Sol: SELECT SUM (rating) FROM Shopkeepers;

⇒ The above query returns a single value which is the sum of all values in attribute rating.

⇒ Output = 36

5) Find the minimum age of Shopkeepers.

Sol: SELECT MIN (age) FROM Shopkeepers;

⇒ The above query returns a single value which is the minimum age from relation Shopkeepers.

⇒ Output = 12.

6) Find the maximum age of Shopkeepers.

Sol: SELECT MAX (age) FROM Shopkeepers;

⇒ The above query returns a single value which is maximum age from relation Shopkeepers.

⇒ Output = 22.

Dealing with NULL values in aggregate functions:

- All aggregate functions except COUNT (*) ignore NULL values in their input collections.
- The count of an empty collection is zero(0).
- All other aggregate functions give NULL value when it is applied to a set which is empty.

Example: Consider a relation R(A, B):

R

A	B
1	6
2	NULL
3	2
NULL	NULL

Table 3.20

1) COUNT (*) = 4
2) COUNT (A) = 3
3) COUNT (B) = 2
4) SUM (A) = 6
5) SUM (B) = 8
6) MAX (A) = 3
7) MAX (B) = 6
8) MIN (A) = 1
9) MIN (B) = 2
10) AVG (A) = 2
11) AVG (B) = 4

Nested queries in SQL:

Definition:

"A nested query is a query that has another query embedded in it; the embedded query is called subquery."

- Mostly, a subquery presents within the WHERE clause of a query.

IN operators:

- The IN operator in SQL is used to check whether a value is in a given set of elements.

Note:
NOT IN is an operator is used to test whether a value is absent in given set of elements.

EXISTS/NOT EXISTS:

- EXISTS used with a subquery.
- It is said to be met when the subquery return at least 1 row.
- NOT EXISTS can be used to test which rows do not exist in a subquery.

ANY/SOME, ALL:

- These operators are used in the Nested subquery to compare sets.
- These operators are combined with (>, <, > =, < =, <>, =)

- "ANY returns true when any of the subquery values meet the condition."
- "ALL returns true when all of the subquery values meet the condition."

Q1 **Following relations are given below:**
Shopkeepers (shid, shname, shrating, shage)
Sales (shid, Iid, Day)
Items (Iid, Iname, Iprice);

Sol: Shopkeepers

Shid	Shname	Shrating	Shrage
1	Rupa	7	20
2	Puja	8	24
3	Sonu	9	23
4	Rani	8	29
5	Mahima	10	40
6	Sonu	7	23

Table 3.21

Sales

Shid	Iid	Day
1	101	10/10/21
2	103	10/10/21
4	101	10/8/21
1	102	18/6/21
2	102	18/5/21
3	104	9/7/21
4	103	9/7/21
5	103	8/7/21
6	101	8/7/21

Table 3.22

Items

Iid	Iname	Iprice
101	ItemA	10K
102	ItemB	20K
103	ItemC	30K
104	ItemA	20K

Table 3.23

Example: Write a SQL query that will find the name of shopkeepers who have sold item 101.
Sol: SELECT sname FROM Shopkeepers WHERE sid IN (SELECT sid FROM sales WHERE bid = 101);
⇒ The nested subquery first gives the set of Shids for shopkeepers who have sold item 101 (the set contains 1, 4 and 6 as Shid), and then the outer (toplevel) query retrieves the names of shopkeepers whose Shid is in this set.
⇒ Output {Rupa, Rani, Sonu}

Example: Write an SQL query to find the names of shopkeepers who have sold item worth 20K.
Sol: SELECT Shname FROM Shopkeepers WHERE Shid IN (SELECT Shid FROM Sales WHERE Iid IN (SELECT Iid FROM Items WHERE Iprice = '20K'));
⇒ The inner subquery finds the set of Iids of items having a price of 20K, which is 102 and 104.
⇒ The one level above subquery finds the set of Shids of shopkeeper's who have sold one of these items. The set contains 1, 2 and 3 as Shid.
⇒ Lastly, the outer top-level query gives the shopkeepers name whose shid is present in this set of shids.
⇒ Output {Rupa, Puja, Sonu}

Example: Write an SQL query to find the shopkeepers name who have not sold an item worth 20K.
Sol: SELECT Shname FROM Shopkeepers WHERE Shid NOT IN (SELECT Shid FROM Sales WHERE Iid IN (SELECT Iid FROM Items WHERE Iprice = '20K'));
⇒ The inner subquery finds the set of Iids having a price of 20K, which is 102 and 104.
⇒ The one level above subquery finds the set of Shid who have sold either of these items. The set contains Shid as {1, 2, 3}.
⇒ Finally, the top-level query will give the name of shopkeepers whose Shid is not present in this Set {1, 2, 3}.
⇒ Output {Rani, Mahima, Sonu}

Correlated nested queries:

Definition:

"Whenever a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query, the two queries are said to be correlated."

Example: Write a query to find the names of shopkeepers who have sold item number 101.
Sol: SELECT S. Shname FROM Shopkeepers S WHERE EXISTS (SELECT * FROM Sales R WHERE R.Iid = 101 AND R.Shid = S.Shid);

⇒ Here, for each shopkeepers in row S, we test whether the set of Sales rows R such that R.Iid = 101 AND S.Shid = R.Shid is nonempty.
⇒ If this is so, shopkeeper S has sold item 101, and we retrieve the name of shopkeepers.
⇒ Output {Rupa, Rani, Sonu}
⇒ It clearly depicts the correlation(dependency) between inner and outer query.

Example: Write an SQL query to find shopkeepers whose rating is better than some shopkeepers whose name is Sonu.
Sol: SELECT S1.Shid FROM Shopkeepers S1 WHERE S1.Shrating > ANY (SELECT S2.Shrating FROM Shopkeepers S2 WHERE S2.Shname = 'Sonu');
⇒ The above query will give shid's of shopkeepers whose rating is better than rating 7 or 9.
⇒ Output of the above query gives Shids 2, 3, 4, and 5.

Note:

- The above query is a correlated subquery because for every shopkeeper in the outer query, we need to run an inner query.

Note:

ANY is similar to SOME keyword in SQL.
Suppose if there are no shopkeepers named Sonu in the above query, then S1.rating > ANY... will return false, and therefore, the query will return an empty set as an answer.
It means the inner subquery must return at least one row to make the comparison true in the case of ANY.

Example: Write an SQL query to find shopkeepers whose rating is better than every shopkeeper named Sonu.

Sol: SELECT S1.Shid FROM Shopkeepers S1 WHERE S1.Shrating > ALL (SELECT S2. Shrating FROM Shopkeepers S2 WHERE S2. Shname = 'Sonu');

⇒ The above query will give Shid of shopkeepers whose rating is better than rating 7 and 9.

⇒ Output of the above query gives Shid 5.

Note:

- If there are no shopkeepers named sonu, then the comparison S1.Shrating > ALL ... is going to be true. In this case, the above query will return the names of all shopkeepers.

Grey Matter Alert!

IN is equivalent to = ANY, and NOT IN is equivalent to <> ALL.

Q1 **Scan the below relational schema:**
Staff (Firstname, Lastname, Sex, Salary, Bno)
Branch (Bname, Bnumber)
Branch_locations (Bnumber, Blocation)

Sol:

Staff

Firstname	Lastname	Sex	Salary	Bno.
Sonu	Sharma	M	30000	5
Anjali	Singh	F	35000	5
Mahima	Seth	F	40000	4
Somya	Jain	F	25000	1
Rahul	Goyal	M	45000	4

Table 3.24

Branch_locations

Bnumber	Blocation
1	Mumbai
5	Chennai
4	Banglore
5	Mumbai

Table 3.25

Branch

Bname	Bnumber
CSIT	1
Chemical	5
Electrical	4

Table 3.26

Example: Write an SQL query to retrieve the First name of staff who works for branches 1, and 4.

Sol: SELECT Fnames FROM Employee WHERE Dno IN (1, 4);

Example: Write an SQL query to find the First name and Last name of the staff who works for the branch located in 'Bangalore'.

Sol: SELECT Firstname, Lastname FROM Staff WHERE Bno IN (SELECT Bnumber FROM Branch_locations WHERE Blocation = 'Bangalore');

Example: Write an SQL query to find out the First names of all the staff where salary is greater than the salary of all staff in branch number 5.

Sol: SELECT Firstname FROM Staff WHERE Salary > ALL (SELECT Salary FROM Staff WHERE Bno = 5);
⇒ Output of this query gives First names = {Mahima, Rahul}

Previous Years' Question

Consider the following relation:
Cinema (theatre, address, capacity)
Which of the following options will be needed at the end of the SQL query?
SELECT P1.address FROM Cinema P1
Such that it always finds the addresses of theatres with maximum capacity.

1) WHERE P1.capacity > = ALL (Select P2 × capacity from Cinema P2)
2) WHERE P1.capacity > = ANY (Select P2 × capacity from Cinema P2)
3) WHERE P1.capacity > ALL (Select max (P2 × capacity) from Cinema P2)
4) WHRE P1.capacity > ANY (Select max (P2 × capacity) from Cinema P2)

Sol: Option 1) **(GATE-2015 Set-3)**

The GROUP BY and HAVING Clauses

Group by:

- GROUP BY clause tells that a SQL SELECT statement can partition result rows into groups depending on their values in one or several columns.
- It is mandatory for all the attributes that are used along with the GROUP BY clause to appear in the SELECT clause.
- If an attribute is not present in the GROUP BY clause, then it must appear only inside the aggregate function in the SELECT clause.

Note:

- GROUP BY clause is often used with COUNT, MIN, MAX, SUM (i.e. with aggregate function)
- If the grouping column contains NULL values, then all NULL values are grouped together.

Having clause:

- SQL provides a HAVING clause, which is often used in conjunction with a GROUP BY clause, to return only those group of tuples which satisfies the provided condition.

Example: Consider the following relation schema:
Employee (Eid, Ename, Sex, Salary, Dno)

The relation given is used to store information about the employees of a company, where Eid is the key and Dno indicates the department to which the employee is assigned.

Sid	Sname	Sex	Salary	Bno
101	Maya	F	20,000	1
102	Rohan	M	30,000	2
103	Kavita	F	28,000	1
104	Rahul	M	24,000	3
105	Manoj	M	30,000	1
106	Supriya	F	10,000	2
107	Rekha	F	32,000	3

Table 3.27

Q1 **Write a SQL query to find the average salary of staff in each branch.**

Sol: SELECT Dno, AVG(Salary) as Avg-Salary FROM Employee GROUP BY Dno;

Q2 **What will be the output of the query 'To find the average salary in each branch'.**

Sol: Output:

Bno	Avg-Salary
1	26,000
2	20,000
3	28,000

Q3 **Write a SQL query to list the branch numbers with an average salary greater than 20,000.**

Sol: SELECT Dno FROM Employee GROUP BY Dno HAVING AVG(Salary) > 17,000;

⇒ This query results Dno = {1, 2} because department numbers 1 and 2 have an average salary greater than 17,000.

Q4 **Consider the following relations:**

Employee

Emp_id	Emp_name
101	Amit
102	Rohan
103	Somya

Performance

Emp_id	Project_code	Rating
101	PA	10
102	PB	9
103	PB	10
102	PA	8
103	PC	5

The SQL query is given below
SELECT E. Emp_name, SUM(P.rating) FROM Employee E, Performance P
WHERE E. Emp_id = P. Emp_id GROUP BY E. Emp_name;
The number of tuples returned by the SQL query is _____

Sol: GROUP BY E. Emp_name means all employee names that are same should be kept in one row. Here, there are 3 employee names, and all are distinct. So, no need to execute the query, and we can tell the number of tuples returned = 3.

OR

Step 1: Perform cross product between Employee E and performance P, we will get 15 rows-relation.

Step 2: Execute WHERE clause; WHERE E. Emp_id = P. Emp_id
Delete rows which does not satisfy WHERE condition.

Step 3: Execute GROUP BY clause: GROUP BY E. Emp_name and then SELECT clause.

Output:

E. Emp_name	SUM(P.rating)
Amit	10
Rohan	17
Somya	15

Rack Your Brain

Consider the relation student with Roll no. as the key

Student

Rollno	Marks
1	92
2	93
3	94
4	NULL

The following SQL query is successfully executed on the relation student.
SELECT avg(Marks) FROM student;
The output of the above query is:

1) 93 **2)** 93.5
3) 94 **4)** NULL

Having and Where clause:

Note:

Generally, HAVING is used in conjunction with GROUP BY, but it is not mandatory.

Example: SELECT Eid, Ename FROM Employee HAVING salary > 20,000;

Note:

If HAVING clause is used without GROUP BY, then it will work the same as the WHERE clause, i.e. we can write the same above query as:
SELECT Sid, Sname FROM staff WHERE salary > 20000;

Q5 **Consider the following relations:**

Shopkeepers

Shid	Shname	Shrating	Shage
22	Komal	7	20
29	Jon	8	22
31	Rahul	9	23
32	Sara	10	21
58	Vikas	5	24
64	Rohan	10	25
71	Preeti	8	22
85	Rahul	7	27
75	Seema	9	28

Output the result of the SQL query specified below:
SELECT S. Shrating, MIN(S. Shage) AS Minage FROM Shopkeepers S WHERE S. Shage > = 23 GROUP BY S. Shrating HAVING COUNT (*) >1;

Sol: The above query will result in a relation containing two attributed rating and minage.

Step 1: Condition in WHERE clause is given as S.Shage > = 23.
It will eliminate all the rows from the Shopkeeper relation which does not satisfy the given condition.

Shrating	Shage
9	23
5	24
10	25
7	27
9	28

(Eliminate all other attributes Shid and Shname as they are not required.)
Step 2: Now, we will apply the GROUP BY clause and sort the table by grouping the attribute rating.

Shrating	Shage
5	24
7	27
9	23
9	28
10	25

Step 3: Finally, we will apply the HAVING clause condition, i.e. COUNT(*)>1.
The groups having rating 5,7 and 10 will be eliminated.
Final answer:

Shrating	Minage
9	23

(As query will result a relation with two attributes rating and minage). Minage will contain 23 corresponding to rating 9 as it is the minimum value among 23 and 28.

Note:
Conditions specified in the WHERE clause are applied before forming the groups and Conditions specified in the HAVING clause are applied only after forming the groups

Order of keywords in SQL:
When we write a SQL query, then the order of keywords that we follow is as follows:

1) SELECT
2) FROM
3) WHERE
4) GROUP BY
5) HAVING
6) ORDER BY

Order of execution in SQL:

1) FROM
2) WHERE
3) GROUP BY
4) HAVING
5) SELECT
6) ORDER BY

The WITH clause:

Definition:

"Using SQL WITH clause, we can give a sub-query block a name (a process also called sub-query refactoring), and later it can be referenced in several places within the main SQL query."

- We can define a temporary view using the WITH clause.
- It is basically a drop-in replacement to the normal sub-query.
- It's not easy to read a query if a nested subquery is used to write it.
- We can write any query using WITH clause in such a way that its logic becomes clearer.
- Also, it allows us to use view definition in multiple places within a query

Example: Consider the relational schemas specified below:
Account (account_number, branch_name, balance)

Account

Account_number	Branch_name	Balance
12345678	Mumbai	25,000
45289243	Hyderabad	30,000
34219341	Bangalore	40,000
98214623	Mumbai	45,000
81243258	Hyderabad	20,000
24319824	Mumbai	20,000
34294526	Banagalore	30,000

Table 3.28

The number of tuples returned by the following SQL query is _____

```
WITH branch_total (branch_name, value) As
        SELECT branch_name, SUM (balance)
        FROM account
        GROUP BY branch_name
WITH branch_total_avg (value) As
        SELECT AVG (value)
        FROM branch-total;

SELECT branch_name FROM branch_total, branch_total_avg WHERE
branch_total.value ≥ branch_total_avg.value;
```

Sol: The first query will return:

branch_total

Branch_name	Value
Bangalore	35,000
Hyderabad	25,000
Mumbai	30,000

Table 3.29

Second query will return: branch_total_avg as average of (35,000, 25,000, 30,000) = 30,000.

Value
30,000

Now, the last query will give the name of the branch where branch-total, value ≥ 30,000

Branch_name
Bangalore

Thus, 1 tuple will be returned.

Joins in SQL:

- To integrate the record sets of two or more relations based on the equality of the common attributes shared by them, the join operation is used.
- Different types of join between relations.
  1) NATURAL JOIN
  2) INNER JOIN
  3) LEFT OUTER JOIN
  4) RIGHT OUTER JOIN
  5) FULL OUTER JOIN

Consider the following relations R and S:

R

A	B
1	a
2	b
3	c
4	d
5	e
6	f

S

A	B
1	g
2	h
3	i
7	j
8	k
9	l

Table 3.30

1) Natural join:

- When we apply natural join on two relations R and S, no need to specify any join condition explicitly.
- Each such pair of common attributes is included only once in the resulting relation.
  Example: SELECT * FROM (R NATURAL JOIN S);
- It will directly join R and S based on attribute A.
- This is the same as query:
  SELECT * FROM R NATURAL JOIN S ON R.A = S.A;

2) Inner join:

- The default type of join in a joined relation.
- Two or more relations are joined based on some join condition on attributes of two relations.

Example: SELECT * FROM R INNER JOIN S ON R.A = S.A;

A	B	A	B
1	a	1	g
2	b	2	h
3	c	3	i

Table 3.31

Consider the relations R and S

R

A	B
1	a
2	b
3	c
4	d
5	e
6	f

S

C	D
1	g
2	h
3	i
7	j
8	k
9	l

Table 3.32

1) SELECT * FROM R INNER JOIN S ON R.A = S.C;
Sol: It will give output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i

Table 3.33

- INNER JOIN can be represented diagrammatically as:

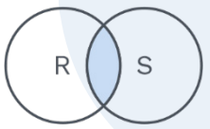


Fig. 3.3

Note:
INNER is a optional keyword. INNER JOIN is same as JOIN.

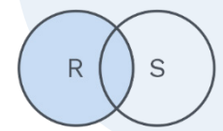
3) left outer join:

- Along with the condition satisfying tuples, all the condition failing records of the left-hand side relation must show up in the resultant relation.
- If the tuple from left-hand side relation that do not match any tuple in right-hand side relation, then NULL values need to be added for the columns of the right relation.

Example: SELECT * FROM R LEFT OUTER JOIN S ON R.A = S.C;
Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	NULL	NULL
6	f	NULL	NULL

Table 3.34



LEFT OUTER JOIN

Fig. 3.4

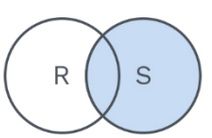
4) Right outer join:

- Every tuple in the right relation must appear in the result.
- If the tuples from the right hand side relation that do not match any tuple in the left-hand side relation, then it is padded with NULL values for the attribute of the right relation.

Example: SELECT * FROM R RIGHT OUTER JOIN S ON R.A = S. C;
Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
NULL	NULL	7	j
NULL	NULL	8	k
NULL	NULL	9	l

Table 3.35



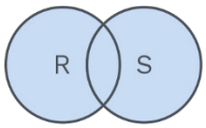
RIGHT OUTER JOIN

Fig. 3.5

5) Full outer join:

- Full outer join includes results of both left and right outer join combined.
- The rows of the left and right-hand side relation that fails to satisfy the JOIN condition are included in the resultant set with NULL values in the right and left-hand relation attributes respectively.

Example: SELECT * FROM R FULL OUTER JOIN S ON R.A = S.C;



FULL OUTER JOIN

Fig. 3.6

Output:

A	B	C	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	NULL	NULL
6	f	NULL	NULL
NULL	NULL	7	j
NULL	NULL	8	k
NULL	NULL	9	l

Table 3.36

Q6 **Consider the following given relations.**

Restaurant

Rest_id	Rest_name	Rest_city
101	Madhulikam	Bangalore
121	Lazzez	Mumbai
114	Sagar Ratna	Delhi
120	Food Plaza	Bangalore
180	Tandoor Palace	Kolkata
135	Foodie	Chennai

Food_item

Food_id	Food_name	Rest_id
22	Sweets	101
24	Dosa-Idli	114
15	Veg Roll	120
18	Biryani	180
20	Pizza	121
14	Burger	121
10	Paneer Roll	120

Consider the SQL query specified below:
SELECT * FROM Restaurant R LEFT OUTER JOIN Food_item F ON R. Rest_id= F. Rest_id;
What will be the number of tuples returned by the given query?

Sol: The above SQL query returns 8 tuples.
The resultant relation is

Rest_id	Rest_name	Rest_city	Food_id	Food_name
101	Madhulikam	Bangalore	22	Sweets
121	Lazeez	Mumbai	20	Pizza
121	Lazeez	Mumbai	14	Burger
114	Sagar Ratna	Delhi	24	Dosa-Idli
120	Food Plaza	Bangalore	10	Paneer Roll
120	Food Plaza	Bangalore	15	Veg Roll
180	Tandoor Palace	Kolkata	18	Biryani
135	Foodie	Chennai	NULL	NULL

Views in SQL:

Definition:

"known as Virtual Tables in SQL. A view in SQL is a single table that is derived from the other tables; Here, these other tables can be base tables or previously defined views."

- A view can be defined using the create view command.
- We need to declare a view name along with the query that computes view.
- Create view command is defined as:
  CREATE VIEW V AS <query expression> where <query expression>.

Example: Consider the following relation schemas:
Staff(Firstname, Lastname, Sex, Salary,Ssn)
Works_for(Cno,Essn, Hours)
Company(Cname, Cnumber,Clocation,)
CREATE VIEW WORKS_FOR1 AS
SELECT Firstname, Lastname, Cname, Hours FROM Staff, Company, Works_for WHERE Cno
= Cnumber AND Ssn = Essn;

- Here, WORKS_FOR1 will take attribute names from given relations (Staff, Works_for, Company) as we have not specified any new attribute names for it.

WORKS_FOR1

Firstname	Lastname	Cname	Hours

We can also explicitly specify the name of attributes of a view.

Schema definition in SQL:

- CREATE TABLE command is used to define a relation.

Syntax: CREATE TABLE <table name> (<column 1> datatype (width), <column 2> datatype (width) ....., <integrity-constraint$_1$>, ..., ... <integrity-constraint$_k$>);

- We have numerous datatypes in SQL
  1) Numeric: Number, Float, int, Real, Decimal
  2) Character: Char, Varchar2, Nchar, Long, Nvarchar2
  3) Date: Date, Time, Interval
  4) Boolean: Boolean

- **Constraints in SQL:**
  Sometimes while creating an SQL relation (table), we need to specify some constraints on attributes.
  These constraints are

1) NOT NULL
2) UNIQUE
3) PRIMARY KEY
4) FOREIGN KEY
5) CHECK
6) DEFAULT

Example: A SQL query having constraints where the requirement is as follows: For a relation department: dept_id should be a primary key, dept_name should be unique, and there should be an attribute location with no constraints. Create a table (relation) department specifying the above requirements.

Sol: Create table Department (dept_id NUMBER PRIMARY KEY, dept_name VARCHAR2(30), location VARCHAR2(50), UNIQUE (dept_name));

Note:
Constraints in SQL can be defined at two levels.
1) Column level **2)** Table level

Constraints

1) **NOT NULL:** NOT NULL is a constraint that is used when an attribute value is not allowed to be NULL.
2) **UNIQUE:** UNIQUE constraint is used when two rows in any relation should not be same.
   Syntax: UNIQUE (Aj_1, Aj_2, ..., Aj_m)
3) **PRIMAR KEY:** PRIMAR KEY (Aj_1, Aj_2, ..., Aj_m) specification says that attributes Aj_1, Aj_2, ..., Aj_m form the primary key for the relation. The primary key attributes have to be UNIQUE and not NULL.
   There will be only one primary key for one table.
4) **FOREIGN KEY:** FOREIGN KEY is used to show the referential integrity. By default, the primary key attribute of a referenced table is referred by a foreign key of another table.
5) **CHECK (P):** The CHECK clause specifies a condition C that needs to be satisfied by every tuple in the relation.
   Whenever a tuple is inserted or modified, the check clause checks the specified condition on them.
6) **DEFAULT:** DEFAULT constraint is used to set a default value for a column.

Grey Matter Alert!

It is always better to mention a primary key for any relation, but it is not mandatory.

Examples:

1) CREATE TABLE Customer (Customer-name char (20), Customer_street char (30), Customer_city char (30), PRIMARY KEY (Customer-name));

2) CREATE TABLE Account (account_number char (10), branch_name char (15), balance integer, PRIMARY KEY (account_number), CHECK (balance > = 0));

INSERT, DELETE and UPDATE Statements in SQL:

These three commands are used to modify the database in SQL.

INSERT Command:

- We use the INSERT command to insert(add) a new tuple to a relation.
- We need to mention the name of the relation as well as the list of attribute-values for the tuple.
- All the attributes should be used with the INSERT command in the same linear manner as in CREATE TABLE command.

Syntax: INSERT INTO <table name> VALUES (attr_value1, attr_value2, ...);

Let us create a table Staff, and then we will insert values in it.

```
CREATE TABLE Employee
(   Fname       Varchar (15)        NOT NULL,
    Lname       Varchar (15)        NOT NULL,
    Ssn         Char (9)            NOT NULL,
    Bdate       Date,
    Sex         Char,
    Salary      Decimal (10, 2),
    Super_Ssn   Char (9),
    Dno         int                 NOT NULL,
PRIMARY KEY (Ssn), FOREIGN KEY (Super_Ssn) REFERENCES Staff (Ssn),
FOREIGN KEY (Dno) REFERENCES Department ((Dnumber));
INSERT INTO Staff VALUES ('Rohan', 'Sahni', '653298653', '1996-10-10', 'M',
37000, '532467821', 4);
```

Note:

- User can also specify explicit attribute names that correspond to the values given in the INSERT command.

Example: Suppose, we want to insert a tuple in a relation Staff(Firstname,Lastname,Deptno, Ssn).

Then, we have to write:

```
INSERT INTO Staff(Firstname,Lastname,Deptno,Ssn)
```

VALUES ('Rohan', 'Sahni', 4, '653298653');

DELETE Command:

- It removes tuple from a relation.
- At once, Rows are deleted from only one table.

Note:

0,1, or more than 1 row can be deleted by a single DELETE command depending on the number of rows selected by the condition specified in the WHERE clause.

Syntax: DELETE FROM <table name> WHERE <condition>

Example: DELETE FROM employee WHERE Lname = 'Sahni'

DELETE FROM employee WHERE Dno = 5

DELETE FROM employee // Deletes all rows from Employee table

UPDATE Command:

- To modify values of attributes of one or more selected tuples, we use UPDATE Command
- In UPDATE Command, a WHERE clause is used, which selects all those tuples we need to modify.
- A SET clause is used in the UPDATE command to mention the attributes to be modified with their new values.

Syntax: UPDATE <table name> SET <column name> = <value expression>{, <column name> = <value expression>} [WHERE <selection condition>];

Example: Let us want to modify the location of Unacademy's project number 20 to 'Chandigarh' as well as controlling department number to '25' in a relation UnacademyProject.

UnacademyProject-schema (Pname, Pnumber, Plocation, Dnum);

SQL Query: UPDATE Project SET Plocation = 'Mumbai', Dnum = 5 WHERE Pnumber = 20;

Schema change statements in SQL:

- There are some commands in SQL that are used to alter a schema.

DROP Command:

- The DROP TABLE command is used to removes a relation from SQL database.
- In other words, The DROP TABLE command deletes all information regarding the relation, which we are considering to drop from the database.

Syntax: DROP TABLE <table name>;

Note:
Difference between command DROP TABLE r and DELETE FROM r:

- DROP TABLE r deletes not only all tuples of relation r, but also the schema for r.
- Once r is dropped, we need to recreate the table using CREATE TABLE Command to insert new rows.
- DELETE FROM r, retains relation r but deletes all tuple in r.

ALTER Command:

- ALTER TABLE command is used for adding attributes to an existing relation.
- **Syntax:** ALTER TABLE <table name> ADD <column name> <column type>
- We can drop attributes from a relation by the command
  ALTER TABLE <table name> DROP <column name>

Rack Your Brain

Which of the following is a data manipulation command?

1) SELECT **2)** GRANT
3) DROP **4)** INSERT

3.2 RELATIONAL ALGEBRA

Introduction:

- Relational algebra is a procedural query language.
- In a procedural language, the user provides a specific procedure to execute the operations on the database to get the desired output.
- Relational algebra employs a set of operations that inputs one or two tables and produces a resultant output relation based on some pre-defined clauses.
- Select, project, union, set difference, cartesian product, and rename are the basic operations that summarises relational algebra.
- There are several other operations such as set intersection, natural join, and division. We can define these operations in terms of the fundamental operations.
- The select, project and rename operations operate on one relation. So, they are known as unary operations.

Grey Matter Alert!

In a non-procedural language, the user doesn't provide any specific procedure to obtain the information but specifies the desired information.

Selection and projection:

- The SELECT operator chooses those tuples in the output that satisfy the specified condition.
- The select operation is denoted by
  $\sigma_{<selection\text{-}condition>}$ (Relation name)
  Consider the relation Sailors:

Eid	Ename	Rating	Age
101	Richa	7	24
105	Rohan	9	20
120	Mahesh	8	26
145	Abhishek	10	29

Table 3.37

To select those tuples of Sailors relation where rating is greater than 8. We can write:

$\sigma_{rating > 8}$ (Sailors)

The output we get the relation shown below:

Sid	Ename	Rating	Age
105	Rohan	9	20
145	Abhishek	10	29

Table 3.38

Example: Consider the relation Employee:

Employee

Eid	Ename	Salary	Dno
1	a	10K	1
2	b	12K	1
3	c	11K	2
4	d	13K	3
5	e	7K	3

Table 3.39

The expression: $\sigma_{salary > 10K \text{ AND } Dno = 3}$ (Employee) evaluates to the relation.

Eid	Ename	Salary	Dno
4	d	13K	3

- Comaprison operators that can be used in selection conditions are: <,<=,=,≠,>=,>.
- The selection operation is applied to each tuple individually.
- The degree (i.e. number of attributes) of the resultant relation after applying the SELECT operation is going to have the same degree as of relation R.
- Number of rows in final output relation £ Number of rows in given input relation(R).
  i.e. $|\sigma_c(R)| = |R|$ where c is any condition.

Note:

- The select operation is commutative i.e.

$$\sigma_{<cond\ 1>}(\sigma_{<cond\ 2>}(R)) = \sigma_{<cond\ 2>}(\sigma_{<cond\ 1>}(R))$$

- We can also write,

$$\sigma_{<cond\ 1>}(\sigma_{<cond\ 2>}(\ldots(\sigma_{<cond\ n>}(R))\ldots)) = \sigma_{<cond\ 1> \text{ AND } \ldots <codn\ 2> \ldots \text{ AND } <cond\ n>}(R)$$

Project operation:

- The project operation is used to choose certain columns from the table and trashes out the other attribute fields.
- The projection operator is denoted by π.
- One can visualise the result of the project operations as an input relation is vertically separated into two relations. One having needed columns (attributes) and the other comprising of the discarded columns.
- The general form to represent the project operation is:

$$\pi_{<attribute\ list>}(Relation)$$

  where <attribute list> is the targeted sub-list of attributes belonging to the relational attribute set.
- The resultant relation after applying the project operation will contain only those attributes that are mentioned in <attribute list> in the same order as they are oriented in the list.

Example: Consider the relation Employee:

Employee

Eid	Ename	Salary	Dno	Sex
101	Raman	30,000	1	M
102	Sneha	20,000	1	F
103	Maya	20,000	2	F
104	Ranjith	20,000	2	M
105	Mahesh	15,000	3	M

Table 3.40

The expression, $\pi_{Salary, Sex}$ (Employee) evaluates to the relation:

Salary	Sex
30,000	M
20,000	F
20,000	M
15,000	M

Table 3.41

The expression $\pi_{Ename, Salary, Sex}$ (Employee) evaluates to the relation:

Ename	Salary	Sex
Raman	30,000	M
Sneha	20,000	F
Maya	20,000	F
Ranjith	20,000	M
Mahesh	15,000	M

Table 3.42

- The project operation results in a set of a distinct tuple as the Project operation removes duplicate tuples.

Note:

- $\pi_{<\text{list 1}>}(\pi_{<\text{list 2}>}(R)) = \pi_{<\text{list 1}>}(R)$
  is true if attributes in <list 1> is also present in <list 2>
- Project operation is not commutative.

Note:

Projection operation in relational algebra is equivalent to SELECT DISTINCT in SQL.

E.g. $\pi_{\text{Sex, Salary}}$ (Employee)
is similar to
SELECT DISTINCT Sex, Salary FROM Employee;

Previous Years' Question

Which of the following query transformations (i.e. replacing the LHS expression by the RHS expression) is incorrect?
R_1 and R_2 are relation, C_1 and C_2 are selection conditions, and A_1 and A_2 are attributes of R_1.

1) $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$ **2)** $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$

3) $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$ **4)** $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$

Sol: Option 4) **(GATE-1998)**

Rename operation:

- Rename operation is used to rename either the relation name or the attribute names or both.
- The rename operator is denoted by rho (ρ).
- The general rename operation when applied to a relation R of degree n is denoted by any of the following three forms:
  $\rho_{s(B_1, B_2, \ldots B_n)}(R)$
  (or) $\rho_s(R)$
  (or) $\rho_{(B_1, B_2, \ldots B_n)}(R)$
  where s is the new relation name, and B_1, B_2, ... B_n are the new attribute names.
- $\rho_{s(B_1, B_2, \ldots B_n)}(R)$ renames both the relation and its attributes.
- $\rho_s(R)$ renames the relation only.

- $\rho_{(B_1, B_2, \ldots B_n)}(R)$ renames the attributes only.

  E.g. Consider the relation customer:

Cid	Cname	Sex
101	S_1	M
102	S_2	F
103	S_3	F

Table 3.43

$\rho_{(Customerid,\ Customername,\ Sex)}$ (customer)

We will get

Customer id	Customer name	Sex
101	S_1	M
102	S_2	F
103	S_3	M

Table 3.44

Set operations:

- The following standard operations on sets are available in relational algebra:
  1) Union ($\cup$)
  2) Intersection ($\cap$)
  3) Set-difference (–)
- There are binary operations and they are applied to two sets.

Grey Matter Alert!

- The two relations on which either union or intersection or set difference operations are implemented upon must necessarily have similar data types of tuples. This condition is called type compatibility.
- Two relations $R(A_1, A_2, \ldots A_n)$ and $Q(B_1, B_2, \ldots B_n)$ are referred to as type compatible if they have the same degree and domain (A_i) is equal to the domain (B_i).
- Type compatible is also called as union compatible.

The union operation:

- P ⋃ Q (P union Q): The resultant relation stores the record set comprising of tuples available in P or Q or both.
- The resultant relation has same schematic representation as that of P.
- The UNION operation does not involve duplicate records in the final relation.

The intersection operation:

- Given two relations R and S, R ⋂ S gives the resultant relation that includes all tuples that are in both relation R and S.
- We will assume that the schema of the resulting relation of R ⋂ S will be the same as schema of R.

The set-difference operation:

- Set-difference, denoted by R-S (R minus S), the result of this relation is a relation that includes all tuples that are present in relation R but not in relation S.
- Here also, we assume that the schema of the resulting relation R-S is same as schema of R.

Note:

- (i) R ⋃ S = S ⋃ R (ii) R ⋂ S = S ⋂ R. It means union and intersection both are commutative operations.
- The set difference operation is not commutative i.e. R – S ≠ S – R.

Example: Consider a relation between employee E_1 and E_2.

E_1

Eid	Ename	Age	Rating
20	Somya	24.0	7
30	Rahul	25.0	8
40	Ranjith	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.45

E_2

Eid	Ename	Age	Rating
30	Rahul	25.0	8
35	Satyam	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.46

I) Now, $E_1 \cup E_2$ will be:

Eid	Ename	Age	Rating
20	Somya	24.0	7
30	Rahul	25.0	8
35	Satyam	24.0	9
40	Ranjith	24.0	9
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.47

II) $E_1 \cap E_2$ will be:

Eid	Ename	Age	Rating
30	Rahul	25.0	8
50	Yashvi	23.0	10
60	Sonam	27.0	8

Table 3.48

III) $E_1 - E_2$ will be:

Eid	Ename	Age	Rating
20	Somya	24.0	7
40	Ranjith	24.0	9

Table 3.49

IV) $E_2 - E_1$ will be:

Eid	Ename	Age	Rating
35	Satyam	24.0	9

Table 3.50

Note:

- Intersection can be expressed in terms of union and set-difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

- Both union and intersection are associative operations.

$$R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \cap (S \cap T) = R \cap (S \cap T)$$

Cartesian product (cross product) operation:

- It is also known as cross join, denoted by ×.
- Cross Product is a binary set operation.
- Relations on which we apply cross product need not be union compatible
- The cartesian product operation, allows us to combine information from any two relations.
- R × S returns a relation whose schema contains all the attributes of R followed by all the attribute of S, i.e. the result of $R(A_1, A_2, ..., A_n)$ × $S(B_1, B_2, .., B_m)$ will result in a relation T with degree n + m attributes. $T(A_1, A_2, ... A_n, B_1, B_2, ... B_m)$ in that order.
- In cross product of R and S, if R has m tuples denoted as |R| = m and S has n tuples denoted as |S| = n then R × S will have m × n tuples.

Note:

The cartesian product operation is mostly useful when followed by a selection.

Example: Consider the relation R and S:

R

A	B	C
1	a	b
2	c	d
3	e	f

S

D	E
1	c
2	d

Table 3.51

(R contains 3 tuples and 3 attributes. S contains 2 tuples and 2 attributes)
Then R × S will be:

A	B	C	D	E
1	a	b	1	c
1	a	b	2	d
2	c	d	1	c
2	c	d	2	d
3	e	f	1	c
3	e	f	2	d

Table 3.52

Containing 3 × 2 = 6 tuples and 3 + 2 = 5 attributes.
The expression $\sigma_{A = D}$ (R × S) will give the result:

A	B	C	D	E
1	a	b	1	c
2	c	d	2	d

Table 3.53

Division operations:

- It is represented by ÷ and is useful for expressing certain kind of queries.
- The Division operation is defined using the basic operators of the algebra.
- Implementation of Division using basic operations:

$R \div S$

Step 1: $T_1 \leftarrow \pi_{(R-S)}(R)$

Step 2: $T_2 \leftarrow \pi_{(R-S)}((S \times T_1) - R)$

Step 3: $T \leftarrow T_1 - T_2$

Example: Consider relation R and S as follows:

R

A	B
a_1	b_1
a_2	b_1
a_1	b_2
a_2	b_2
a_1	b_3

S

A
a_1
a_2

Table 3.54

Then $R \div S$ will be : $T \neg R \leftarrow S$

T

B
b_1
b_2

Step 1: $T_1 \leftarrow \pi_{(R-S)}(R) = T_1 \leftarrow \pi_B R = T_1$

B
b_1
b_2
b_3

Table 3.55

Step 2: (a) $S \times T_1 =$

A	B
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3

Table 3.56

(b) $(S \times T_1) - R$

A	B
a_2	b_3

$T_2 \leftarrow \pi_{(R-S)}((S \times T_1) - R)$ will give: T_2

B
b_3

Step 3: $T \leftarrow T_1 - T_2$

T_2

B
b_1
b_2

Table 3.57

Rename operation:

- In relational algebra, we do not have any name for the results through which we can refer them.
- Most of the time result of a relational algebraic expression usually takes name of field from the original relation.

Previous Years' Question

Consider a database that has the relation schema CR(StudentName, CourseName). An instance of the schema CR is as given below:

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

- $T_1 \leftarrow \pi_{CourseName}\ (\sigma_{StudentName\ =\ SA}\ (CR))$
- $T_2 \leftarrow CR\ ,\ T_1$

The number of rows in T_2 is _____

Sol: Range 4 to 4 **(GATE Set-1-2017)**

- But, It is always good to give them names, using the rename operator denoted by ρ.
- Given a relational-algebra expression E, the expression ρ_x (E) returns the result of expression E under the name x.
- Another form of the rename operation is as follows:

$$\rho_{x(A1, A2, \ldots An)} (E)$$

returns the result of expression E under the name x and with attributes renamed to $A_1, A_2, \ldots, A_n$.

Example: Rename the attributes Empno, Empname of relation employee to Eno, Ename.

$$\rho_{(Eno, Ename)} (Employee)$$

A complete set of relational algebra operations:

- $\{\sigma, \pi, U, -, \times\}$ is a complete set of relational algebra operations.
- It means using a sequence of operations from this set, any other relational algebra operations can be expressed.

Example: Intersection operation can be expressed using union and set-difference as follows.

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Division operation can be implemented using p, x and – (set-difference)
Similarly, a join operation can be implemented using x(cartesian product) and select operations.

$$R \bowtie_{<condition>} S \equiv \sigma_{<condition>} (R \times S)$$

3.3 JOINS

- The join operations is one of the most useful operations in relational algebra.
- The join operation denoted by $\bowtie$, is used to combine related tuples from two relations.
- Join operation is used more frequently than cross-product even though a join is a cross product followed by a selection.
- Reason behind it is cross-product produces a larger results compare to result produce by join.
- Thus, join is very important relational operation.

Condition joins:

- Conditional join operation takes a pair of relation instances as arguments and a join condition c and returns a new relation.
- It is defined as:

$$R \bowtie_c S = \sigma_c (R \times S)$$

Example: Consider the relation Employee E and Project P.

E

Eid	Ename	Rating	Age
20	Ravish	8	24.0
30	Radha	7	25.0
40	Shyam	9	26.0

P

Eid	Pid	Day
20	120	2/8/21
40	115	5/6/21

Table 3.58

The result of $E \bowtie_{E.Eid < P.Eid} P$ is

Eid	Ename	Rating	Age	Eid	Pid	Day
20	Ravish	8	24.0	40	115	5/6/21
30	Radha	7	25.0	40	115	5/6/21

Table 3.59

Equijoin:

- Equijoin involves join condition with equality comparisons.
- Equijoin is a join where = comparison operator is used.

Note:

In the result of an Equijoin, we always have one or more pairs of attribute that have identical values in every tuples.

Consider the relation Employee and Drives as follows:

Employee

Eno	Ename	Sex	Age
101	Ravi	M	24.0
102	Satyam	M	25.0
103	Meera	F	23.0
104	Rohan	M	27.0

Drives

EmpNo	Car
101	Volvo
103	Mercedes
104	Jaguar
104	Toyota

Table 3.60

The result of Employee $\bowtie_{\text{Employee.Eno = Drives.EmpNo}}$ Drives

Eno	Ename	Sex	Age	Emp No	Car
101	Ravi	M	24.0	101	Volvo
103	Meera	F	23.0	103	Mercedes
104	Rohan	M	27.0	104	Jaguar
104	Rohan	M	27.0	104	Toyota

Table 3.61

Natural join:

- Natural join is denoted by *.
- Natural join is used to combine two relation R and S having a common attribute(s) names.
- If we apply Natural join on two relations R and S, then it is denoted as R * S.
- We don't need to write equality conditions explicitly when two relations are joined using natural join.

Note:

The standard definition of natural join requires that the two join attributes should have the same name in both relations.
If not, then a renaming operation is applied first before applying Natural join.

Natural join returns similar attributes only once in the resulting relation. Consider the relation between Student and Performance.

Student

Rollno	Student_name
1	Amit
2	Priya
3	Rohan
4	Komal

Performance

Rollno	Subject_code	Marks
1	A	84
1	B	90
2	C	92
3	A	85

Table 3.62

The result of stu_per ← student * performance

Stu_per

Rollno	Student_name	Subject_code	Marks
1	Amit	A	84
1	Amit	B	90
2	Priya	C	92
3	Rohan	A	85

Table 3.63

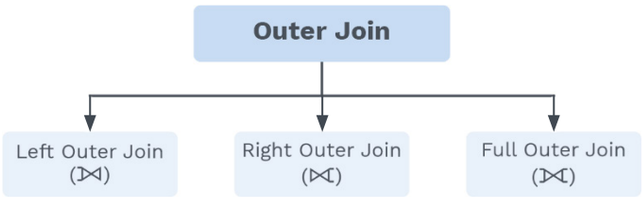
Note:

Condition join, Equijoin and Natural join are called Inner join.

Outer-join:

- It is a variation of JOIN that efficiently deals with missing data.
- Here, since the schema of the result includes all attributes from both relations (table), if a tuple from the first relation does not match tuple in the second relation, we simply put NULL values in the tuple of the resulting relation on the attributes of the second relation.

- There are three forms of outer join:



1) **Left outer join (⟕):** The left outer join takes all tuples in the left relation that do not match with any tuple in the right relation, and padded these tuples with NULL values for all the attributes of the right relation. Add them to the result of inner join.
2) **Right outer join (⟖):** It considers the records of the right-hand side relations that fails to follow the JOIN condition. It pads NULL values in the left-hand-side attributes for those records.
3) **Full outer join (⟗):** In full outer join, along with the result of the inner join of two relation, it will contain padding tuples from both left and right relation that does not match with any tuples from other (right and left relation respectively), relations.

E.g. Consider the two relations R and S as follows:

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S

D	E
a_1	e_1
a_3	e_2
a_4	e_3

Table 3.64

i) The result of R $⟕_{A=D}$ S will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_2	b_2	c_2	NULL	NULL
a_3	b_3	c_3	a_3	e_2

Table 3.65

ii) The result of R $⟖_{A=D}$ S will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_3	b_3	c_3	a_3	e_2
NULL	NULL	NULL	a_4	e_3

Table 3.66

iii) The result of the full outer join $R \overset{\leftrightarrow}{\bowtie}_{A=D} S$ will be

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_3	b_3	c_3	a_3	e_2
a_2	b_2	c_2	NULL	NULL
NULL	NULL	NULL	a_4	e_3

Table 3.67

Note:

1) $R \bowtie S \subseteq R \overset{\leftrightarrow}{\bowtie} S$

2) $R \bowtie S \subseteq R \overset{\leftarrow}{\bowtie} S$

3) $R \overset{\leftrightarrow}{\bowtie} S = R \overset{\leftarrow}{\bowtie} S \cup R \overset{\rightarrow}{\bowtie} S$

Previous Years' Question

Let R and S be two relations with the following schema.

$$R(P, Q, R_1, R_2, R_3)$$

$$S(P, Q, S_1, S_2)$$

Where {P, Q} is the key for both schemas. Which of the following queries are equivalent?

I. $\pi_P (R \bowtie S)$

II. $\pi_P (R) \bowtie \pi_P (S)$

III. $\pi_P (\pi_{P,Q} (R) \cap \pi_{P,Q} (S))$

IV. $\pi_P (\pi_{P,Q} (R) - (\pi_{P,Q} (R) - \pi_{P,Q} (S)))$

1) Only I and II **2)** Only I and III
3) Only I, II and III **4)** Only I, III and IV

Sol: Option 4) **(GATE-2008)**

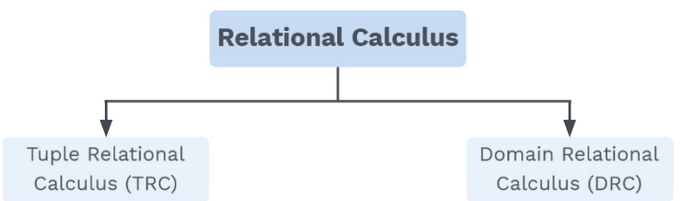
Extended relational algebra operations:

- In basic relational algebra, we don't have some features like arithmetic operations, string concatenation etc.
- Later, the basic relational algebra has been extended.
- Example – To allow arithmetic operations as a part of the projection, to allow aggregate operations etc.
- We can extend the projection operation by allowing arithmetic functions to be used in the projection list.

E.g. $\pi_{id, name, dept\_name, salary\ ,\ 12}$ (Instructor)

Relational calculus:

- Relational calculus is an alternative to relational algebra.
- Relational calculus is considered to be non-procedural language.



- Relational algebra, tuple relational calculus and domain relational calculus are equivalent in power (expressive power of the languages is identical).
- Thus, relational algebra and relational calculus are relationally complete.

Tuple relational calculus:

- In TRC, we specify the tuple variables.
- A tuple relational calculus query is {t|COND (t)}
  where t is a tuple variable and COND(t) is a conditional (Boolean) expression that describes t.
- **Example:** Shopkeeper(Firstname, Lastname, Rating)
  To find all shopkeepers whose rating is more than 8. It can be represented in TRC as:
  {t | Shopkeeper(t) AND t.rating > 8}
  It will give all the tuples from shopkeeper relation that satisfies t.rating > 8 conditions.
  When we want to list only some attributes from Shopkeeper relation ,we can write
  the following expression.
  {t.Firstname, t.Lastname | Shopkeeper(t) AND t.rating > 8 }
- A formula (Boolean condition) in TRC expression is made up of one or more atoms connected using the logical operators AND, OR and NOT.
- A formula is recursively defined as one of the following:
  I) Any atomic formula
  II) If p and q are formulas then so are
  $\neg p$, $p \wedge q$, $p \vee q$, $p \Rightarrow q$.
  III) $\exists t\ (p(t))$, where t is a tuple variable.
  IV) $\forall t\ (p(t))$, where t is a tuple variable.

Note:

$\exists$ represents existential quantifier
$\forall$ represents universal quantifier

Free and bounded variables:

- $\exists$ and $\forall$ are used to **bind** the variable.
- In a formula, if it does not contain an occurrence of a quantifier that binds, the variable then a variable is said to be free.

Example:
Consider the following relational schema:
Employee (Fname, Lname, Ssn, Sex, Salary, Bdate, Address, Super_Ssn, Dno)
Department (Dname, Dnumber, Mgr_Ssn, Mgr_start_date)
Dept_locations (Dnumber, Dlocation)
UnacademyProject (UPname, UPnumber, UPlocation, UDnum)
Works_on (Essn, Pno, Hours)
Dependent (Essn, Dependent_name, Sex, Bdate, Relationship)

Q1 **Project out the name and address information of the department number ten employees.**

Sol: $\{e \times Fname, e \times Address \mid Employee(e) \wedge e \times Dno = 10\}$

- Only the free tuple variables in a TRC expression need to appear to the left of the vertical bar(|).
- Whatever we want to display should be written left to the vertical bar.
- Here, if a tuple satisfies the conditions specified after the bar then the attributes Fnames and Address are retrieved for each such tuple.

Q2 **List out the names of the dependent(s) of the employee with first name as 'Ram'.**

Sol: $\{d.Dependent\_name \mid Dependent\ (d) \wedge (\exists e)\ (Employee\ (e) \wedge e.Ssn = d.Essn \wedge e.Fname = \text{'Ram'})\}$

- Here, d is free tuple variable, tuple variable e is bound to the existential quantifier.
- We are performing join between two relation Employee and Dependent and then selecting those combinations of tuples that satisfy the condition.
- Tuple variable d ranges over Dependent relation and tuple variable e ranges over Employee relations.

Q3 **Retrieve the first names and addreses of the 'Research' department employees.**

Sol: {e.Fname, e.Address | Employee (e) AND (∃d) (Department(d) ∧ e.Dno= d.Dnumber ∧ d.Dname = 'Research'}

- Tuple variable e ranges over the relation Employee and, tuple variable d ranges over the relation Department.

Q4 **Retrieve the project number, authority department number and respective Ssn of the manager for all such projects having the base location in 'Mumbai'.**

Sol: {p.Pnumber, p.Dnum, d.Mgr_Ssn | Project (p) AND Department (d) ∧ p.location = 'Mumbai' ∧ p.Dnum = d.Dnumber}

Q5 **Retrieve the project number, authority department number and respective last-name, DOB and address of the manager for all such projects having base location in 'Mumbai'.**

Sol: {p.Pnumber, p.Dnum, e.Lname, e.Bdate, e..Address | Project (p) ∧ Employee (e) ∧ p×location = 'Mumbai' ∧ (∃d) (Department (d) ∧ p.Dnum = d×Dnumber ∧ d×Mgr_Ssn = e×Ssn)}

- When there are two or more free tuple variables, take cartesian product of those relations. Out of all possible combinations of tuples, only the combinations that satisfy the condition are selected.

Q6 **Retrieve the project number in which people with teh last name 'Kohli' is either a worker or a manager in control of the concerned department.**

Sol: {p.Pnumber | Project (p) AND (((∃e) (∃w) (Employee (e) AND works-on (w) AND w.Pno = p.Pnumber AND e.Lname = 'Kohli' AND e.Ssn = w.Essn)) OR ((∃m) (∃d) (Employee (m) AND Department (d) AND p.Dnum = d.Dnumber AND d.Mgr_Ssn= m.Ssn AND m.Lname = 'Kohli'))))}

Q7 **List out the first names of the employees working on projects under department number 7.**

Sol: {e.Fname | Employee (e) AND ((∃(x)) (∃(w)) (project(x) AND Works_on(w) AND x.Dnum = 7 AND w.Essn = e.Ssn AND x.Pnumber = w.Pno))}

Note:

1) $P \rightarrow Q \equiv \neg PVQ$

2) $\neg(\forall x(p(x))) \equiv \exists x(\neg p(x))$

3) $(\forall x)\ (p(x)) \equiv \neg(\exists x)\ (\neg p(x))$

4) $(\exists x)\ (p(x)) \equiv \neg(\forall x)\ (\neg p(x))$

Q8 **Project out the employees first names with no dependents.**

Sol: {e.Fname | Employee (e) AND (NOT (∃d) (Dependent(d) AND e.Ssn = d.Essn))}

Rack Your Brain

Which of the following relational algebra expression is equivalent to the given tuple calculus expression.
{t | t ≡ r ∧ (t[M] = 25 ∧ t[N] = 40)}

1) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

2) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

3) $\sigma_{(M = 25)}(r) - \sigma_{(N = 40)}(r)$

4) None of these

Unsafe relational calculus expression:

- The expression is known to be a safe expression if it is guaranteed to yield a finite number of tuples as its result.
- If relational calculus is not yielding finite number of tuples as its result then expression is unsafe.

E.g. {e | ¬ employee(e)}
There are infinitely many tuples that does not belong to employee relation.
So, the above example is unsafe.

Previous Years' Question

Which of the relational calculus expression is not safe?

1) $\{t \mid \exists u \equiv R_1 (t[A] = u[A]) \wedge \neg \exists s \equiv R_2 (t[A] = s[A])\}$

2) $\{t \mid \forall u \equiv R_1 (u[A] = \forall x \forall \Rightarrow \exists s \equiv R_2 (t[A] = s[A] \wedge s[A] = u[A]))\}$

3) $\{t \mid \neg (t \equiv R_1)\}$

4) $\{t \mid \exists u \equiv R_1 (t[A] = u[A]) \wedge \exists s \equiv R_2 (t[A] = s[A])\}$

Sol: Option 3) **(GATE-2001)**

Note:

For every safe Tuple relational calculus query there is an equivalent relational algebra query.

Domain relational calculus:

- Another type of relational calculus is domain relational calculus.
- Domain relational calculus uses Domain variables.
- Variable that ranges over the values in the domain of some attributes is known as domain Variable.
- Domain relational calculus is equivalent in power to tuple relational calculus.
- A Domain relational calculus query is of the form

$$\{x_1, x_2, x_3, \ldots, x_n \mid P(x_1, x_2, \ldots, x_n)\}$$

Where $x_1, x_2, \ldots, x_n$ are domain variables and P represents a formula which is made up of atoms. An atom in the domain relational calculus is of the form:

1) An atom of the form $R(x_1, x_2, \ldots x_n)$ where R is a relation name.

Note:

To make a domain calculus expression more brief, we can write

$$\{x_1, x_2, x_3, \ldots x_n \mid R(x_1\ x_2\ x_3) \text{ AND } \ldots\}$$

Instead of $\{x_1, x_2, x_3, \ldots xn \mid R(x_1, x_2, x_3)$ AND ...$\}$
by removing commas in a list of variables.

2) An atom of the form x op y where x and y are domain variable and op is a comparison operator {<, ≤, =, >, ≥, ≠}.

3) An atom of the form x op c where x is a domain variable and op comparison operator and c is a constant.

- Domain relational calculus formulae is build up from atoms using following rules:
- Domain relational calculus formulae is build up from atoms similar to tuple calculus relation

Example: Consider the relation schemas:
Employee (Fname, Lname, Ssn, Sex, Salary, Bdate, Address, Super_Ssn, Dno)
Department (Dname, Dnumber, Mgr_Ssn, Mgr_start_date)
Project (Pname, Pnumber, Plocation, Dnum)
Works-on (Essn, Pno, Hours)
Dependent (Essn, Dependent × name, sex, Bdate, relationship)

Q1 **List the salary as well as address of the employee named 'Somya Jain'.**

Sol: $\text{Employee}\left(\overset{m}{\text{Fname}}, \overset{n}{\text{Lname}}, \overset{o}{\text{Ssn}}, \overset{p}{\text{Sex}}, \overset{q}{\text{Salary}}, \overset{r}{\text{Bdate}}, \overset{s}{\text{Address}}, \overset{t}{\text{Super\_Ssn}}, \overset{u}{\text{Dno}},\right)$

- We need 9 variables for employee relation, ranging over each of the domains of attributes of employee in order.
- {q, s | (∃m) (∃n) (∃o) (∃p) (∃r) (∃t) (∃u) (Employee (mnopqrstu) ∧ m = 'Somya' ∧ n = 'Jain')}
- q and s are free variable, as they appear to the left of the vertical bar and not bounded to any quantifier.
- For convenience purpose, we generally quantify only those
- variables that appears in the condition.

Q2 **List the first name, last name and address of all employees who work for the department 'Research'.**

Sol: {m, n, s | (∃g) (∃h) (∃u) (Employee (mnopqrstu) ∧ Department (ghij) ∧ g = 'Research' ∧ h = u)}

$\text{Employee}\left(\overset{m}{\text{Fname}}, \overset{n}{\text{Lname}}, \overset{o}{\text{Ssn}}, \overset{p}{\text{Sex}}, \overset{q}{\text{Salary}}, \overset{r}{\text{Bdate}}, \overset{s}{\text{Address}}, \overset{t}{\text{Super\_Ssn}}, \overset{u}{\text{Dno}},\right)$

$$\text{Department}\left(\underset{g}{\text{Dname}}, \underset{h}{\text{Dnumber}}, \underset{i}{\text{Mgr\_Ssn}}, \underset{j}{\text{Mgr\_start\_date}}\right)$$

- Here, we are joining two relation employee and department using h = u where h is a domain variable ranges over attribute Dnumber of Department and u is a domain variable ranges over attribute Dno of relation Employee.

Q3 **For every project located in 'Mumbai', list the project number, the controlling department number and department manager's surname and address**

Sol: {x, z, n, s | (∃y) (∃h) (∃i) (∃o) (Project (wxyz) ∧ Employee (mnopqrstu) ∧ Department (ghij) ∧ h = z ∧ i = o ∧ y = 'Mumbai')}

$$\text{Employee}\left(\underset{m}{\text{Fname}}, \underset{n}{\text{Lname}}, \underset{o}{\text{Ssn}}, \underset{p}{\text{Sex}}, \underset{q}{\text{Salary}}, \underset{r}{\text{Bdate}}, \underset{s}{\text{Address}}, \underset{t}{\text{Super\_Ssn}}, \underset{u}{\text{Dno}},\right)$$

$$\text{Department}\left(\underset{g}{\text{Dname}}, \underset{h}{\text{Dnumber}}, \underset{i}{\text{Mgr\_Ssn}}, \underset{j}{\text{Mgr\_start\_date}}\right)$$

$$\text{Project}\left(\underset{w}{\text{Pname}}, \underset{x}{\text{Pnumber}}, \underset{y}{\text{Plocation}}, \underset{z}{\text{Dnum}}\right)$$

Q4 **Find the names of employees who haven't any dependents.**

Sol: {m, n | ∃(o) (Employee (mnopqrstu) ∧ (Not (∃a) (Dependent (abcde) ∧ o = a)))}

$$\text{Employee}\left(\underset{m}{\text{Fname}}, \underset{n}{\text{Lname}}, \underset{o}{\text{Ssn}}, \underset{p}{\text{Sex}}, \underset{q}{\text{Salary}}, \underset{r}{\text{Bdate}}, \underset{s}{\text{Address}}, \underset{t}{\text{Super\_Ssn}}, \underset{u}{\text{Dno}},\right)$$

$$\text{Department}\left(\underset{a}{\text{Essn}}, \underset{b}{\text{Dependent\_name}}, \underset{c}{\text{Sex}}, \underset{d}{\text{Bdate}}, \underset{e}{\text{Relationship}}\right)$$

Previous Years' Question

What is the optimized version of the relation algebra expression $\pi_A\left(\pi_A\left(\sigma_F\left(\sigma_F\left(r\right)\right)\right)\right)$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expression based on the attributes in r?

1) $\pi_{A_1}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

2) $\pi_{A_1}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

3) $\pi_{A_2}\left(\sigma_{(F_1 \wedge F_2)}(r)\right)$

4) $\pi_{A_2}\left(\sigma_{(F_1 \vee F_2)}(r)\right)$

Sol: Option 1) **(GATE Set-3-2014)**

Chapter Summary

- **SQL :** SQL stands for Structured Query Language. It is used to add, delete, modify or obtain data.
- Types of SQL commands:
  DDL, DML, DQL, DCL
- SELECT Clause: SELECT clause is used to list all the desired attributes in the result of a query.
- A simple form of SQL query:

$$\text{SELECT } A_1, A_2, \ldots, A_n$$

$$\text{FROM } r_1, r_2, \ldots, r_m$$

$$\text{WHERE <Condition>;}$$

- **LIKE :** It is a operator that is used for pattern matching of strings.
- **Aggregate function in SQL:**
  SQL provides 5 inbuilt aggregate functions.
  1) Average : AVG **2)** Minimum : MIN
  3) Maximum : MAX **4)** Total : SUM
  5) Count : COUNT
- **Correlated nested query:** Two queries are said to be correlated when a condition in the WHERE clause of the inner query captures and utilizes some fields of the table name specified in the outer query.
- **Joins :** It is used to combine two or more relation based on common attributes between them.
- **Types of Joins:**
  1) Natural Join **2)** Inner Join
  3) Left Outer Join **4)** Right Outer Join
  5) Full Outer Join
- **Relational algebra:** It is a procedural query language.
- Fundamental operation on relational algebra:
- Selection(s)
- Projection(p)
- Union(U)
- Set-difference(-)
- Cartesian Product (x)
- Renaming(ρ)

4 Transaction and Concurrency Control Techniques

4.1 INTRODUCTION

Definition

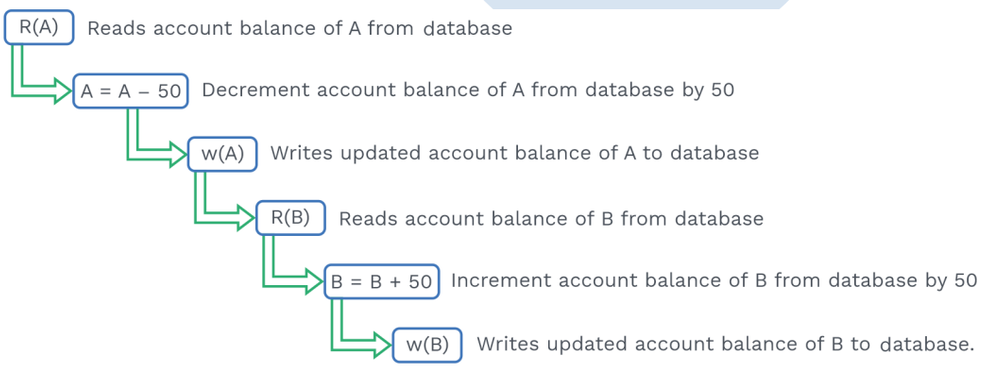
'A transaction is a collection of operations that forms a single logical unit of work'.

For example, if we transfer money from one account to another account, then the transaction consists of two updates one to each account.

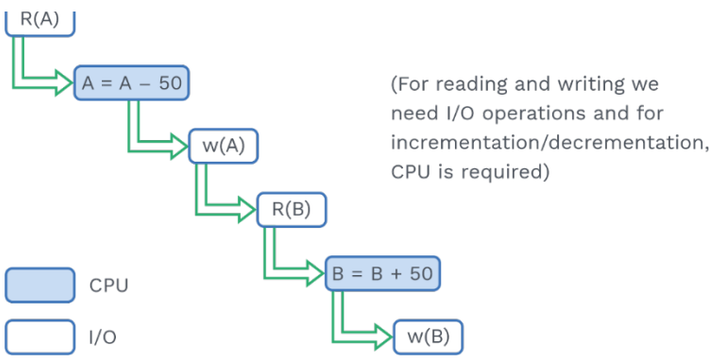
- Consider an example:

Initial account balance of A = 100
Initial account balance of B = 200
Suppose A wishes to transfer ₹ 50 to B.
After transaction account balance of A = 50
After transaction account balance of B = 250

- Let us look at the various steps involved in this single transaction:



- Out of these various steps certain steps require I/O (input-output) operations to be done and certain steps require CPU operations to be done.



- To increase the efficiency of a CPU, it should not remain idle. We can increase the efficiency by assigning CPU to other processes for CPU operations while the CPU is idle.

T_1	I/O	CPU	I/O	I/O	CPU	I/O
T_2	CPU	I/O	CPU	CPU	I/O	CPU

- In this case CPU is not left idle because when transaction T_1 is performing I/O operation, than T_2 is utilising the CPU and when transaction T_2 is performing the I/O operation transaction T_1 is performing CPU operation.

ACID properties

There are 4 ACID properties that a transaction needs to follow:

1)	A	⟶	Atomicity
2)	C	⟶	Consistency
3)	I	⟶	Isolation
4)	D	⟶	Durability

1) Atomicity

Definition

'Atomicity states that either all transactions must reflect properly in the database or none'.

- It means no transaction executes partially.
- Transaction control manager → responsible to ensure atomicity.

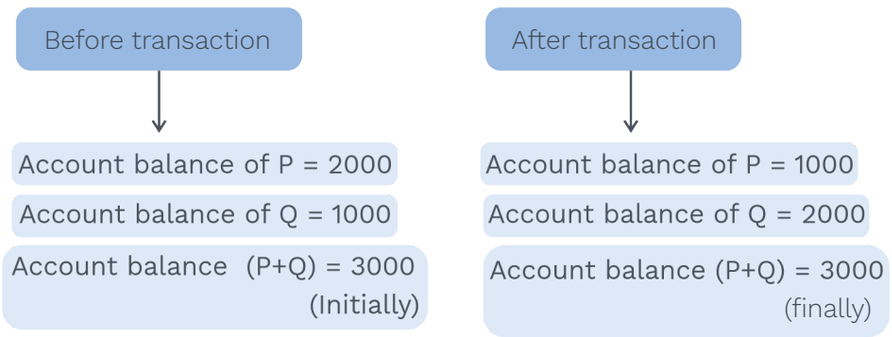
2) Consistency

Definition

'This property ensures that integrity constraints are maintained'.

- In other words, consistency must be there before as well as after the transaction.
- DBMS and application programmer are responsible to ensure consistency of database.

For example, transfer of ₹ 1000 from account P to account Q.



3) Isolation

Definition

'Isolation means each transaction should feel like it is executing alone in the system'.

- The transaction should not feel as if any other transaction is also executing parallelly.

Note:

After executing all the transactions concurrently; the result achieved by the system should be the same as transactions are executing serially (one after the other).

- Concurrency control manager → responsible to ensures isolation.

4) Durability

Definition

This property ensures that if any failure occur (power failure, hard disk crash, etc.) than also system should be able to recover, i.e. even after the failure result of the committed transaction remains same.

- Durability says that whatever changes we are making is permanent, and in future, even there is any failure, these changes will never be lost.

Previous Years' Question

Which of the following is NOT a part of the ACID properties of database transactions?

1) Atomicity **2)** Consistency
3) Isolation **4)** Deadlock-freedom

Sol: 4) **(GATE-2016, Set-01)**

Types of failures

- There are many kinds of failures that can occur in a system.
- Some failures do not result in loss of information but some failures result in loss of information.

1) Transaction failure: Logical error and system error are two types of errors that lead a transaction to fail.
- Example of logical errors are overflow, resource limit exceeded, etc.
- Deadlock is a system error. As in this case, system enters an undesirable state.
- Thus, a transaction can't continue its normal execution.

2) System crash: Due to the hardware malfunction, a database leads to the loss of content, and therefore transactions might halt.

3) Disk failure: Due to the head crash or failure during a data transfer operation, the content of a disk block might be lost.

Transaction states

- A transaction passes through several phases in its life-cycle.

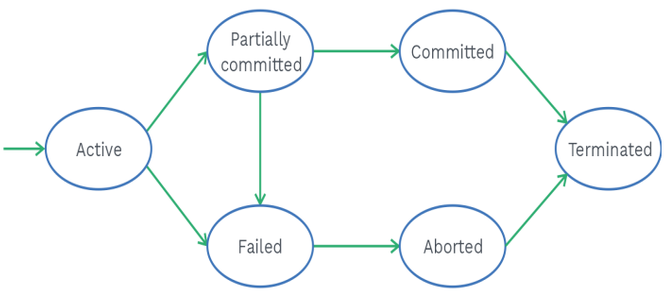


Fig. 4.1 Life Stages of a Transaction

Active state

- It is an initial state of a transaction.

Partially committed state

- A transaction is in a partially committed state when the final operation of a transaction is executed. From here, it is a chance that a transaction might be aborted.

Failed state

- When a transaction discovers that it cannot proceed with its normal execution, it moves to the failed state.

Aborted state

- If a transaction faces roll back, all the changes impacted by the transaction are altered to form the previous image of the database. Consequently, the transaction enters into the aborted life stage.

Committed state

- When a transaction completes its execution successfully, it moves to the committed state.

Note:

Hey learners!!
Do you know what a terminated transaction is?
Well, a terminated transaction is the one that has either been committed or aborted.

- A system has two choices when it moves to the aborted state:
  1) Restart the transaction
  2) Kill the transaction
- Transaction will restart only when the transaction was aborted due to any hardware or software error.

Concurrent executions

- Concurrent execution allows more than one transaction to run concurrently in a transactional system.
- So, inconsistency may be arises when multiple transactions are allowed to update data concurrently.

Why do we need concurrent executions?

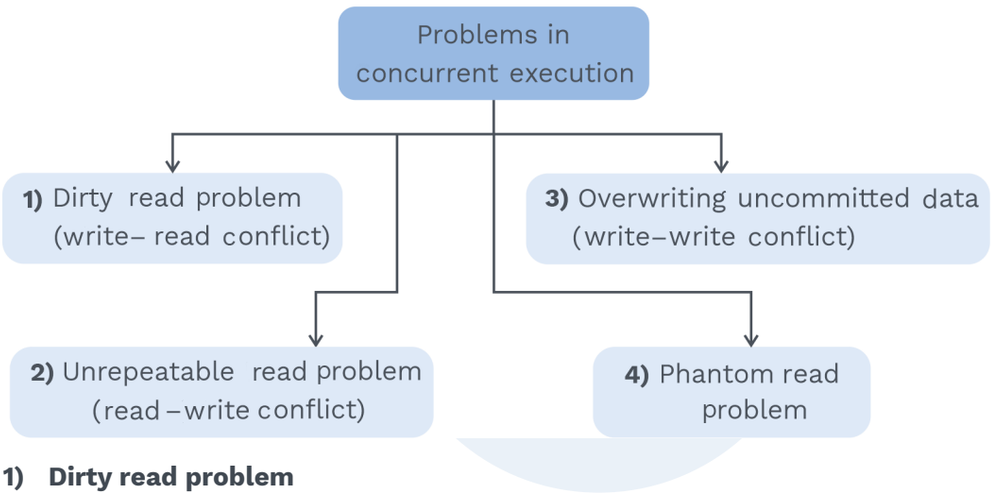
The reasons to allow concurrency are:

1) It enhances database performance and reduces waiting.
2) Better throughput and utilisation of resources can be accomplished.

Problem due to concurrent execution of transactions

- Due to the interleaving of operations between transactions, database can lead to an inconsistent state.

4.2 PROBLEMS IN CONCURRENT EXECUTION



1) Dirty read problem

Definition

'Reading the data written by an uncommitted transaction is called as a dirty read'.

- Let us make an assumption that a transaction T1 modifies a data item X. Another transaction T2 undergoes read(X). But, due to any reason transaction T1 is failed and the value of the database item is rolled back to the old value.
- So, the value read by T2 is the incorrect one.

Note:

The write-read conflict is also known as reading uncommitted data.

E.g.

	T_1 (A sending ₹ 50 to B)	T_2 (A is incrementing its balance by 4%)

ROLL BACK ↑	R(A) // 100
	A = A – 50 // 50
	w(A) // 50	

		R(A) // 50
		X = A * 0.04
		A = A + X // 52
		w(A) // 52

FAIL	R(B) B = B + 50 w(B)	

When transaction T_1 fails and T_2 executes completely, it will rollback and change the value of A back to its old value.

- So, because transaction T_1 is not completed and committed and T_2 is reading the data written by an uncommitted transaction, T_2 has read the incorrect value of A.

Note:

- Dirty read does not implies inconsistency.
- It only creates problem when the uncommitted transaction fails and rollbacks due to any reason.

2) Unrepeatable read problem (read-write conflicts)

- Assume a scenario where T_1 performs two simultaneous read operations on a specific data item. Another transaction T_2 modifies the content of the data item in the time gap amidst the two read operations.
- Consequently, T_1 will obtain distinct values at different time.

A = 20000 (Initially).

T_1	T_2
R(A) // 20000	
	R(A) // 20000
	A = A – 15000//5000
	w(A)
R(A) // 5000	
A = A – 10000	
w(A)	

- In the above example, when T_1 reads the value of A for the second time, it will read a different value of A because the transaction T_2 has changed the A's value in between first and second read of T_1.
- A real life example of unrepeatable read problems can be considered as:

During a flight reservation, let's say a customer C_1 checks the seat availability and tries to book a ticket meanwhile suppose another customer C_2 booked the ticket so, when customer C_1 reaches to the payment page, it might happen that C_1 will get to read a different value of seat availability.

3) Overwriting uncommitted data (write-write conflicts)

- Write-write conflict problem arises when an update performed by a transaction (say T_1) on a data item is lost due to the update done by another transaction (say T_2).

For example,

Initially A = 100

	T_1 (A Sending ₹ 50 to B)	T_2 (A is incremented its balance by 4%)
	R(A) // 100	
	A = A – 50 // 50	
		R(A) // 100
		X = A * 0.04
		A = A + X // 104
This update is lost ←	w(A) // 50	
		w(A) // 104
	R(B)	
	B = B + 50	
	w(B)	

- This problem is also known as lost-update problem.
- Consider T_1 and T_2 are two transactions executed almost at the same time, and it is already shown in the above figure that operations of these transaction are interleaved.
- So, the final value of item A is not going to be correct because transaction T_2 reads the value of A before it is changed by T_1 in the database.
- This results in loss of the value that is updated by T_1.

$A = 100 \rightarrow A = 50 \rightarrow A = 104$

4) Phantom read problem

- A transaction T_1 reads a set of tuples from a table satisfying the condition given in an SQL query.
- Now, consider another transaction T_2, which inserts a new tuple into the table satisfying the same condition given in the same SQL query present in the transaction T_1.
- T_1 repeats the same query again, then T_1 will result in a row that is not present previously (phantom).
- This situation is known as the phantom read problem.

For example,

T_1	T_2
Eno | Ename | Salary 1 | Asha | 5000 3 | Kimi | 4000 SELECT * FROM Employee WHERE Salary > = 3000;	
	INSERT INTO Employee VALUES (4, Disha, 3500);
Eno | Ename | Salary 1 | Asha | 5000 3 | Kimi | 4000 4 | Disha | 3500 SELECT * FROM Employee WHERE Salary > = 3000;	

(Here we can see SELECT is executed twice, the second time we get additional row.)

Grey Matter Alert!

Hey learners!!
Do you have any idea about the incorrect summary problem?
Now, we will study the incorrect summary problem with an example.

- Suppose one transaction calculating an aggregate summary on different database items.
- While some of these items are updated by other transactions, incorrect summary results can reflect as some values might be calculated by aggregate function before updating their values and some after updating their values.

  For example, let's say we wish to sum up the values of K, X and Y.

  K = 50, X = 100, Y = 200

T_1	T_2	
	Sum = 0	
	R (K)	
	Sum = Sum + K	
R (X)		
X = X + 500		
w (X)		
	R (X)	
	Sum = Sum + X	
	R (Y)	→ T_2 reads X after 500 is added and reads Y before 200 is added, so a wrong summary is a result.
	Sum = Sum + Y	
R (Y)		
Y = Y + 200		
w (Y)		

The sum of K = 50, X = 100, Y = 200 supposed to be 350. But after updating of values of X sum will be (50 + 600 + 200) = 850, if again T_1 tries to increase values of Y by 200 summary will be different.

- Activities like when a transaction starts, ends, aborts or commits must be tracked by the system so that recovery can be possible.
- Following operations are tracked by the recovery manager:

1) Begin transaction
2) Read or write
3) End transaction
4) Commit transaction
5) Rollback (OR) abort

Commit

Commit transaction indicates a successful end of the transaction and whatever updates done by the transaction is committed to the database and later cannot be undone.

Abort (OR) rollback

Abort (OR) rollback indicates that the transaction has ended unsuccessfully. Thus, whatever changes is done by the transaction to the database must be undone.

- The system keeps a log to monitor all the transaction operations so that later it can retrieve all those information in case of any failure.

Commit point of a transaction

- A transaction is said to be reached its commit point if all the operations of a transaction execute successfully and are reflected in the log.

Rack Your Brain

T_1	T_2
R(y)	
	R(x) R(y) y = x + y w(y)
R(y)	

The following given schedule is suffering from:

1) Lost update problem **2)** Unrepeatable read problem
3) Both 1) and 2) **4)** Neither 1) nor 2)

4.3 SCHEDULE

Definition

'When transactions are executing concurrently in an interleaved manner, then the order of execution of operations from any of the various transaction is known as a schedule'.

- We can represent a schedule consisting of n transactions as $T_1, T_2, \ldots, T_n$.
- Interleaving of operations between the transaction is allowed in any schedule.

Note:

Suppose there are two transactions T_1 and T_2 that have to perform m and n number of operations respectively, then total number of possible schedules $= \frac{(m+n)!}{m!\,n!}$

Note:

We will use shorthand notation R, W, C and A for the operation read_item, write_item, commit and abort, respectively, in any schedule.

Serial schedule

Definition

'If the operations of each transaction are executed consecutively, then the schedule is known as serial schedule'.

Problem with serial schedule

- The main problem with serial schedule is that it restricts the interleaving of operations and thus also limits concurrency.
- The serial schedule also leads to the wastage of processing time of the CPU.
- It also starves transactions. For example, if a transaction T_1 is quite long, another transaction has to wait for T_1 to complete its operations.
- Thus, the serial schedule is not good in practice.

Note:

In the serial schedule, transaction should be either T_1 followed by T_2 or T_2 followed by T_1.

Non-serial schedule

Definition

'A schedule S is non-serial, if the operations of each transaction T participating in the schedule, are interleaved or not executed consecutively'.

Note:

- At a time only one transaction is active in serial schedule.
- In serial schedule, next transaction is performed only when an active transaction is committed.
- 'There are n! different valid serial schedules are possible for a set of n transactions'.
- Consistency is always guaranteed if transaction is executing serially.

Rack Your Brain

Find the number of serial schedules possible when three transactions executing?

Example of serial and non-serial schedule

- Let us consider the current values of accounts A and B are ₹ 1500 and ₹ 2500, respectively.
- Consider the following schedule where T_1 is followed T_2:

T_1	T_2
R(A) A = A – 50 w(A) R(B) B = B + 50 w(B)	
	R(A) X = A * 0.2 A = A – X w(A) R(B) B = B + X w(B)

- The final value of accounts A and B, after the execution is ₹ 1160 and ₹ 2840, respectively.
- Hence, total amount of money in accounts A and B remain same after the execution of both transactions.
- Another serial schedule is also possible, i.e. T_2 followed by T_1.

T_1	T_2
	R(A) X = A * 0.2 A = A – X w(A) R(B) B = B + X w(B)
R(A) A = A – 50 w(A) R(B) B = B + 50 w(B)	

- In this case also, the sum A + B is same and the final values in the accounts A and B are ₹ 1150 and ₹ 2850, respectively.
- But consider an another non-serial schedule as shown below.

T_1	T_2
R(A)	
A = A – 50	
	R(A)
	X = A * 0.2
	A = A – X
	w(A)
	R(B)
w(A)	
R(B)	
B = B + 50	
w(B)	
	B = B + X ⟶ This will not preserve consistency
	w(B)

- After the execution of this schedule, we will get final values of accounts A and B are ₹ 1450 and ₹ 2790 respectively.
- Thus, the sum of amount in account A and B is going to increase by ₹ 240 and the final state is inconsistent.

Note:

- It is the responsibility of concurrency control component of a database system to make sure that after the execution of any schedule, the database should remain in consistent state.

Complete Schedule

Definition

'A schedule S of n transactions T_1, T_2, ..., T_n is said to be a complete schedule if the following condition hold:
The last operation of each transaction is either commit or abort operation'.

Example:

T_1	T_2
R (A)	
	R (A)
A = A – 50	
w(A)	
Commit	
	A = A + 50
	w(A)
	Abort

4.4 SERIALIZABILITY

Introduction:

Definition

'A property that tells about the correctness of the schedule when a concurrent transactions are executing'.

Uses of serializability

To maintain the data item in a consistent state.

Serializable schedule

Definition

'A transaction schedule is serializable if its outcome is equal to the outcome of its transactions executed serially'.

Note:

- A schedule that is not equivalent to any one of the serial schedule is known as a non-serial schedule.

Result equivalent schedule

Two schedules are known as result equivalent if the final state of the database comes out to be the same after execution.

Example

Assume initially,

X=2

Y=5

Given below are two schedules. Detect if they are equivalent or not with regard to the results.

T_1	T_2
R(X)//2	
X = X + 5//7	
w(X)	
R(Y)//5	
Y = Y + 5//10	
w(Y)	
	R(X)//7
	X = X * 3//21
	w(X)

Schedule S1

Final value of
X = 21
Y = 10

T_2	T_2
R(X)//2	
X = X + 5//7	
w(X)	
	R(X)//7
	X = X * 3//21
	w(X)
R(Y)//5	
Y = Y + 5//10	
w(Y)	

Schedule S2

Final value of
X = 21
Y = 10

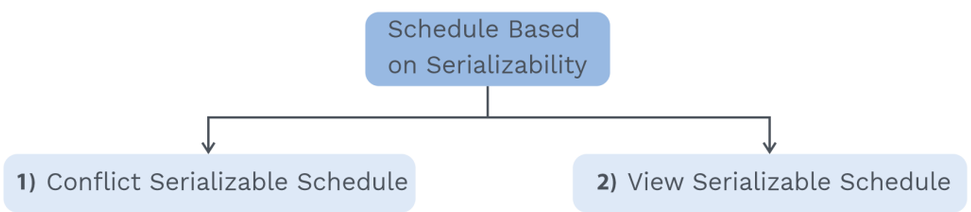
T_1	T_2
	R(X)//2
	X = X * 3//6
	w(X)
R(X)//6	
X = X + 5//11	
w(X)	
R(Y)//5	
Y = Y + 5//10	
w(Y)	

Schedule S3

Final value of
X = 11
Y = 10

Thus schedule S1 which is serial schedule produces the output X = 21 and Y = 10 and non-serial schedule S2 also produces the final output same as S1, i.e.
X = 21 and Y = 10, thus both are result equivalent schedule but final output of S3 is different. It produces output X = 11 and Y = 10, thus S3 is not result equivalent to S1 or S2.

Based on serializability



Conflict serializability

- Let us consider a schedule S where I_i and I_j are the two consecutive operations belongs to transactions T_i and T_j respectively. Here ($i \neq j$).
- Suppose operation I_i and I_j is referring to different data items, then I_i and I_j can be swapped without affecting the outcomes of any instruction in given schedule.

Note:

W(A) denotes write operation on item A.

- When two operations refer to the same data item, order of those operation matters.
- Conflict operations possible are:

	T_i		T_j
	R(A)	.....	W(A)
	W(A)	.....	R(A)
	W(A)	.....	W(A)
Operation	R(A)	.....	R(A) is non-conflicting

Conflict equivalent

Definition

'Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules'.

- Two operations op_1 and op_2 are conflict operations if
  - **i)** op_1 and op_2 are atomic operations of distinct transactions.
  - **ii)** op_1 and op_2 are operated on the same data item.
  - **iii)** Either of op_1 or op_2 must be a write.
- Conflict equivalence establishes conflict serializability.
- 'A schedule S is known to be conflict serializable if it is conflict equivalent to one of the serial schedule'.

E.g.

T_1	T_2
$R_1(A)$	
$w_1(A)$	
	$R_2(A)$
	$w_2(A)$
$R_1(B)$	
$w_1(B)$	

Schedule S1
(non-serial schedule)

T_1	T_2
$R_1(A)$	
$w_1(A)$	
$R_1(B)$	
$w_1(B)$	
	$R_2(A)$
	$w_2(A)$

Schedule S2
(serial schedule)

Conflicts: $R_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow R_2(A)$
$w_1(A) \rightarrow w_2(A)$

Conflicts: $R_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow R_2(A)$
$w_1(A) \rightarrow w_2(A)$

Therefore S_1 is conflict equivalent to S_2.

SOLVED EXAMPLES

Q1 **Identify the given schedules are conflict equivalent (or) not?**
S_1: $R_1(A)$ $w_1(A)$ $R_2(A)$ $w_2(A)$ $R_1(B)$ $w_1(B)$ $R_2(B)$ $w_2(B)$
S_2: $R_1(A)$ $w_1(A)$ $R_1(B)$ $w_1(B)$ $R_2(A)$ $w_2(A)$ $R_2(B)$ $w_2(B)$

Sol:

S_1

T_1	T_2
$R_1(A)$	
$w_1(A)$	
	$R_2(A)$
	$w_2(A)$
$R_1(B)$	
$w_1(B)$	
	$R_2(B)$
	$w_2(B)$

S_2

T_1	T_2
$R_1(A)$	
$w_1(A)$	
$R_1(B)$	
$w_1(B)$	
	$R_2(A)$
	$w_2(A)$
	$R_2(B)$
	$w_2(B)$

Conflicts operations in S_1:
$R_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow R_2(A)$
$R_1(B) \rightarrow w_2(B)$
$w_1(B) \rightarrow w_2(B)$
$w_1(B) \rightarrow R_2(B)$

Conflicts operations in S_2:
$R_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow w_2(A)$
$w_1(A) \rightarrow R_2(A)$
$R_1(B) \rightarrow w_2(B)$
$w_1(B) \rightarrow w_2(B)$
$w_1(B) \rightarrow R_2(B)$

- Here, as we can see all the conflicts are occurring in the same order in both schedules are same.
- Thus, schedule S_1 which is a non-serial schedule is conflict equivalent to serial schedule S_2.
- Therefore, schedule S_1 is a conflict serializable schedule.

Consider the transactions T_1, T_2 and T_3 and the schedules S_1 and S_2 given below:

Previous Years' Question

T_1: $r_1(x)$; $r_1(z)$; $w_1(x)$; $w_1(z)$
T_2: $r_2(x)$; $r_2(z)$; $w_2(z)$
T_3: $r_3(x)$; $r_3(x)$; $w_3(y)$
S_1: $r_1(x)$; $r_3(y)$; $r_3(x)$; $r_2(y)$; $r_2(z)$; $w_3(y)$; $w_2(z)$; $r_1(z)$; $w_1(x)$; $w_1(z)$
S_2: $r_1(x)$; $r_3(y)$; $r_2(y)$; $r_3(x)$; $r_1(z)$; $r_2(z)$; $w_3(y)$; $w_1(x)$; $w_2(z)$; $w_1(z)$
Which of the following statements about the schedules is TRUE?

1) Only S_1 is conflict-serializable
2) Only S_2 is conflict-serializable
3) Both S_1 and S_2 are conflict serializable
4) Neither S_1 nor S_2 is conflict serializable

Sol: 1) **(GATE-2014, Set-03)**

How to test conflict serializability for a given schedule

- Precedence graph helps to ascertain if a particular schedule is conflict serializable or not.
- A precedence graph is based on read and write operations.

Note:

Hey learners!!
Do you have any idea about the precedence graph (Serialization graph)?
Let us read the definition of the precedence graph.

- Definition: 'Precedence graph (Serialization graph) is a directed graph G = (V, E) that consists of a set of nodes V = {T_1, T_2, ..., T_n} and a set of directed edges E = {e_1, e_2, ..., e_m}'.

Algorithm

Following are the steps to draw serialization (precedence) graph:

1)	For all the transactions present in schedule S, create a node as T_i.
2)	Make an edge ($T_i \longrightarrow T_j$) if there is any conflict operation such as write–read (OR) read–write (OR) write–write from transactions T_i to T_j.

- If there is no cycle present in the precedence graph, it means the schedule is conflict serializable.

Note:

Hey Learners!!
The schedule S may or may not be conflict serializable if cycle is present in the precedence graph.

Grey Matter Alert!

($T_i \rightarrow T_k$), ($T_k \rightarrow T_p$), ($T_p \rightarrow T_j$), ($T_j \rightarrow T_i$) is a cycle in a directed graph.

Note:

- Topological sorting is used to obtain the serializability order of any transaction.
- We can obtain more than one possible linear order using topological sorting.

Example: Identify if the given non-serial schedule S is conflict serializable?

T_1	T_2
R(A) w(A)	
	R(A) w(A)
R(B) w(B)	
	R(B) w(B)

Schedule S

Sol: We will check whether this non-serial schedule is conflict serializable or not using precedence graph.

I) Transactions T_1 and T_2 represents node of precedence graph.

II) Conflict operations are represented by edges.
The conflicting operations are:
$R_1(A) \rightarrow W_2(A)$
$W_1(A) \rightarrow R_2(A)$
$W_1(A) \rightarrow W_2(A)$
$R_1(B) \rightarrow W_2(B)$
$W_1(B) \rightarrow W_2(B)$
Which are from transactions T_1 to T_2. There are no conflicts from T_2 to T_1.

So, serialization graph has no cycle. Thus, the given non-serial schedule S is conflict serializable.

SOLVED EXAMPLES

Q2 **Identify the following schedule is conflict serializable or not?**

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

Sol: For the provided schedule, the precedence graph is computed. It will determine if that schedule is conflict serializable or not.

1) Transactions T_1, T_2 and T_3 represent the node of the graph.

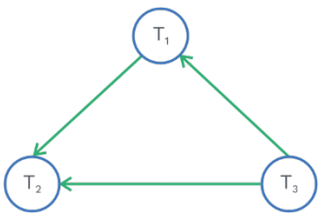
2) Conflict operations:

$R_1(X) \rightarrow W_2(X)$
$W_1(X) \rightarrow R_2(X)$ } Edge from T_1 to T_2
$W_1(X) \rightarrow W_2(X)$

$R_3(X) \rightarrow W_2(X) \longrightarrow$ Edge from T_3 to T_2

$R_3(X) \rightarrow W_1(X) \longrightarrow$ Edge from T_3 to T_1

So, the precedence graph will be:



There is non-existence of a cycle in the above serialization graph, we obtained. So, the given schedule is conflict serializable. Thus, the possible serial schedule that follows from the topological sorting of the obtained serialization graph:
$T_3 \rightarrow T_1 \rightarrow T_2$

Q3 **Test whether the following schedule is conflict serializable or not.**

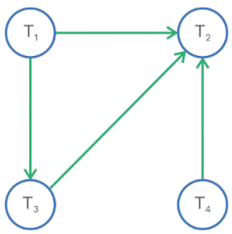
T_1	T_2	T_3	T_4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		

Sol: We will first check using precedence graph whether the given schedule is conflict serializable or not.

1) T_1, T_2, T_3, T_4 represents node of the graph.

2) Conflict operations are represented by edges:
$W_1(B) \rightarrow W_2(B)$: Edges from T_1 to T_2
$R_3(A) \rightarrow W_2(A)$: Edges from T_3 to T_2
$R_4(A) \rightarrow W_2(A)$: Edges from T_4 to T_2
$R_3(B) \rightarrow W_2(B)$: Edges from T_3 to T_2
$W_1(B) \rightarrow R_3(B)$: Edges from T_1 to T_3

So, we will get precedence graph as:



The above graph contains no cycle. Therefore, Conflict serializable.
Possible Serialized schedule after applying topological sort:

1. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$ 2. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$ 3. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

Previous Years' Question

Let $r_i(z)$ and $w_i(z)$ denote read and write operations, respectively, on a data item z by a transaction T_i.
Consider the following two schedules:
S_1: $r_1(x)$ $r_1(y)$ $r_2(x)$ $r_2(y)$ $w_2(y)$ $w_1(x)$
S_2: $r_1(x)$ $r_2(x)$ $r_2(y)$ $w_2(y)$ $r_1(y)$ $w_1(x)$
Which one of the following options is correct?

1) S_1 is conflict serializable and S_2 is not conflict serializable.
2) S_1 is not conflict serializable and S_2 is conflict serializable.
3) Both S_1 and S_2 are conflict serializable.
4) Neither S_1 nor S_2 is conflict serializable.

Sol: 2) **(GATE-CSE-2021, Set-01)**

View serializability
Consider two schedules S1 and S2 having the same set of operation.

1)	If T_i reads initial value of A in S_1, then T_i should also read initial value of data item A in S_2.
2)	If T_J performs final write operation on data item A in schedule S_1 then T_j should also perform the final write operation on A in schedule S_2.
3)	If T_j reads the value produced by T_i in schedule S_1 then T_j must also read the value produced by T_i schedule S_2.

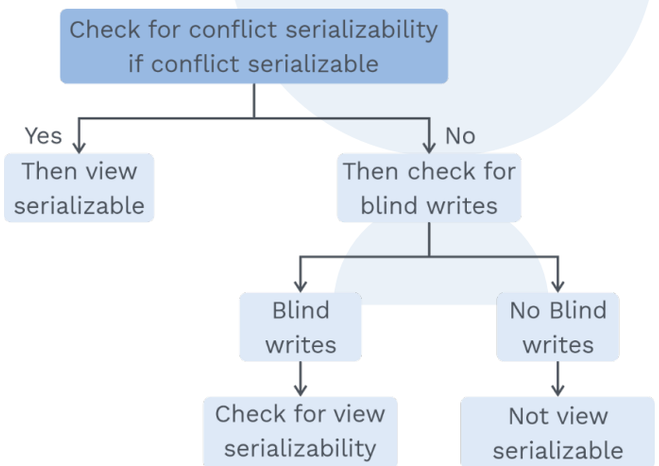
Note:

When a transaction performs a write operation on any data item (say A) without any prior read. It is known as blind write.

Note:

The condition for a schedule to achieve view serializability is: It should be viewed equivalent to either of the realizable serial schedules.

E.g. Consider three transactions: T_1: $r_1(A)$; $w_1(A)$; T_2: $w_2(A)$; and T_3: $w_3(A)$; The schedule S: $r_1(A)$; $w_2(A)$; $w_1(A)$; $w_3(A)$; C_1 ; C_2 ; C_3 ; where blind writes are $W_2(A)$ and $W_3(A)$.



Note:

i) View serializability is a necessary condition for a schedule to be conflict serializable but opposite is not true.

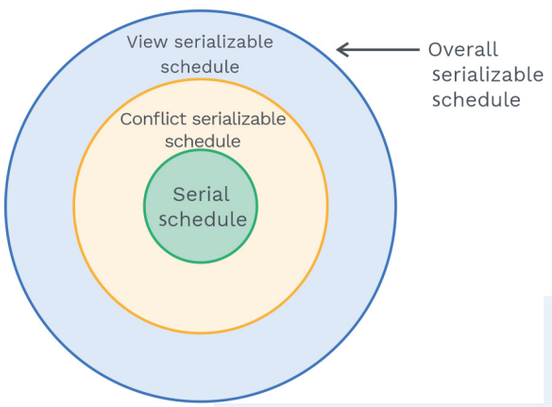


Fig. 4.2 Diagrammatic Representation of Conflict Serializable and View Serializable Schedule

Example: Determine if the following schedules are view equivalent or not.

S_1

T_1	T_2
R(A)	
A = A + 10	
W(A)	
R(B)	
B = B + 20	
W(B)	
	R(A)
	A = A + 10
	W(A)
	R(B)
	B = B * 1.1
	W(B)

S_2

T_1	T_2
R(A)	
A = A + 10	
W(A)	
	R(A)
	A = A + 10
	W(A)
R(B)	
B = B + 20	
W(B)	
	R(B)
	B = B * 1.1
	W(B)

i) If T_1 undergoes initial read operation on any data item X in S_1, it should definitely read the initial value of X in S_2 also.

ii) $W_1(A) \rightarrow R_2(A)$
$W_1(B) \rightarrow R_2(B)$ holds for both schedule.

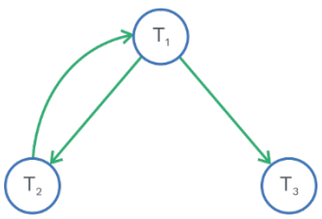
iii) In either of the schedules, final write on data item A and B is performed by T_2. Thus, schedule S_2 is view serializable to S_1.

SOLVED EXAMPLES

Q1 **Check whether the given schedule is view serializable or not.**

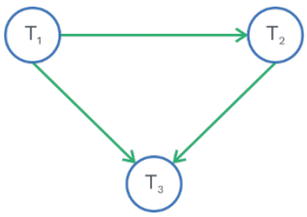
T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

Sol: To find out whether a given schedule is view serializable or not, first, we will find out whether a given schedule is conflict serializable or not, because we know if a schedule is conflict serializable then schedule is also view serializable.
Conflict serializability check:



- The above graph contains cycle, thus it is not conflict serializable.
- Check if there is any blind write, if no blind write then it is not view serializable schedule.
- In the given schedule blind write is present.
- So, now we will test for view serializability for that we will draw polygraph.
- To draw polygraph, following steps are required:
  i) T_1, T_2, T_3 represents nodes of the graph.
  ii) T_1 will execute first, as T_2 is depending on T_1 and T_3 is depending on T_1.

- T_3 should be the last one to execute, based on the writes.
- We will get the following graph:



So, the serial schedule we will get is: $T_1 \rightarrow T_2 \rightarrow T_3$.
(Topological order from topological sort.)
Thus the given schedule is view serializable.

Q2 **Consider the following given schedule:**

T_1	T_2
R(A)	
	W(A)
W(A)	

Is it view serializable schedule?

Sol: First, we will find out whether a given schedule is conflict serializable or not. For that we will draw precedence graph.

- A cycle is present in the above graph. So, it is not conflict serializable.
- Check if there is any blind write, if no blind write then it is not view serializable schedule.
- In the given schedule blind write is present.
- Now, we will test for view serializable, so we will draw polygraph.

T_1 has to start first as read operation is performed by T_1 and since final write operation is also performed by T_1. So, there will be an edge from T_2 to T_1. It means graph is having cycle.
$\therefore$ We can conclude that the given schedule is neither conflict serializable nor view serializable.

Q3 **Is the following given schedule view serializable?**
T_1: R(x), T_2: R(y), T_1: W(x), T_2: R(y), T_3: W(y), T_1: W(x), T_2: R(y)

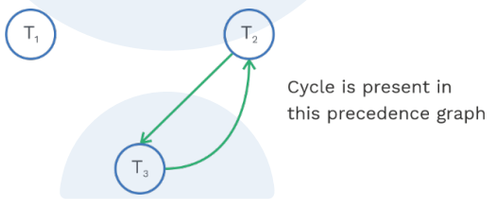
Sol:

T_1	T_2	T_3
R(x)		
	R(y)	
W(x)		
	R(y)	
		W(y)
W(x)		
	R(y)	

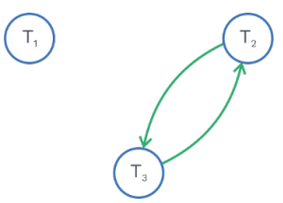
First, we will draw a precedence graph to check conflict serializable schedule:

I) T_1, T_2, T_3 represents nodes of a graph.

II) All conflict operations represent edges of precedence graph.



- Thus, the given schedule is not a conflict serializable schedule.
- We will check if there is any blind write, if no blind write then it is not view serializable schedule.
- In the given schedule blind write is present.
- Now, draw polygraph to check view serializability



As T_1 is the only transaction that performs read and write on variable x so the initial we are ignoring T_1.

T_2 has to start first as initial read operation is performed by T_2 on variable y, there will be an edge from T_2 to T_3.

Due to updated read dependency from T_3 to T_2, there will be an edge from T_3 to T_2.

So, there is cycle in the graph. So, the given schedule is not view serializable.

Rack Your Brain

How many conflicting operations does schedule S have _____?
Schedule S: $r_1(M)$ $w_2(M)$ $r_2(M)$ $w_1(N)$ $r_2(N)$ $w_2(N)$

4.5 BASED ON RECOVERABILITY

- To ensure the atomicity of the transaction, we undo the effect of the transaction whenever it fails.
- Also if a transaction T_j is depending on T_i that needs to be aborted.
- We need to put some restrictions on the schedule to achieve this.

Irrecoverable schedule

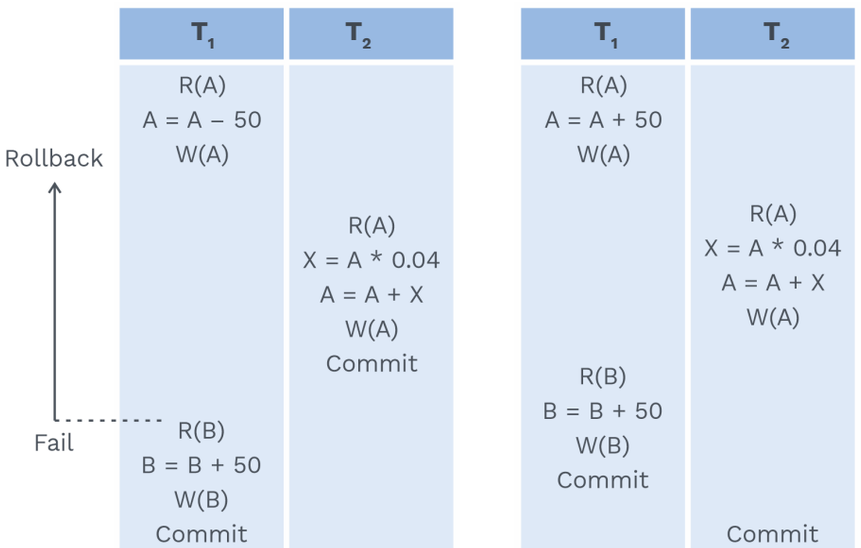
T_1	T_2
R(A)	
W(A)	
	R(A)
	Commit
R(B)	

- The above given schedule is not recoverable (irrecoverable) schedule since commit(T_2) happens before commit(T_1).
- Now suppose T_1 fails before it commits. T_2 has to undergo abort.
- But abort(T_2) is not possible as it has already undergone commit operation.
- So, it is a situation where it is not possible to recover from the failure of T_1.

Recoverable schedules

'A schedule where for each pair of transactions T_i and T_j the commit operation of T_i must appear before the commit operation of T_j if T_j reads a data item previously written by T_i'.

- Given below is an example depicting instances of recoverable and irrecoverable schedules:



Note:

Hey Learners!!
Do you know what a recoverable schedule guarantees?
Well, it guarantees that a transaction does not need to roll back once it commits.

Ensuring recoverability of schedule

T_2 should commit only after T_1 commits if T_2 depends on T_1.
If the above condition is satisfied then it is guaranteed to have a recoverable schedule.

Example: The below given schedule is an example of recoverable schedule.

T_1	T_2
W(A)	
	R(A)
Commit	
	Commit

Note:

A recoverable schedule may have dirty read problem.

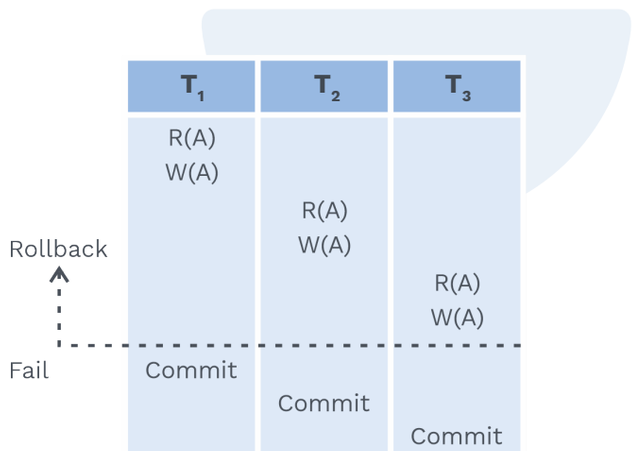
Note:

Set of the recoverable schedules are subset of set of all possible schedules.

Cascading Aborts

If one transaction failure causes multiple transactions to roll back, it is called as cascading roll back (or) cascading aborts.

Example:



- Here, rollback of T_1 will result in rollback of T_2 and rollback of T_2 will result in rollback of T_3 which is cascading rollbacks.
- It is recoverable schedule but due to cascading rollbacks, other problem arises like less throughput, more waiting time.
- Just because of rollback of T_1 here T_2, T_3, T_4 all needs to rollback.
- Thus, cascading rollbacks must be avoided.

Cascadeless schedule

Cascading rollback is not desirable as they undo a good amount of work.
So, it is better if we can avoid the cascading rollback.
Schedule that is restricted in such a way that cascading rollback cannot occur is known as cascadeless schedules.

Definition

'A cascadeless schedule is one, where for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j'.

Note:

Hey learners!!
Do you know that a cascadeless schedule is a recoverable one?
See the below example.

Example:

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

→ Cascadeless schedule

Strict schedule

'Strict schedules are those where a value written by a transaction cannot be read or written by another transaction until the previous transaction commits (or) aborts'.

Example:

S :

T_1	T_2
R(A) W(A) Commit	
	W(A)/R(A)

Here, S is a strict schedule, i.e. it is both cascadeless as well as recoverable.

Note:

Set of all cascadeless schedule are subset of set of recoverable schedule.

Note:

Set of all strict schedule are subset of set of all cascadeless schedule.

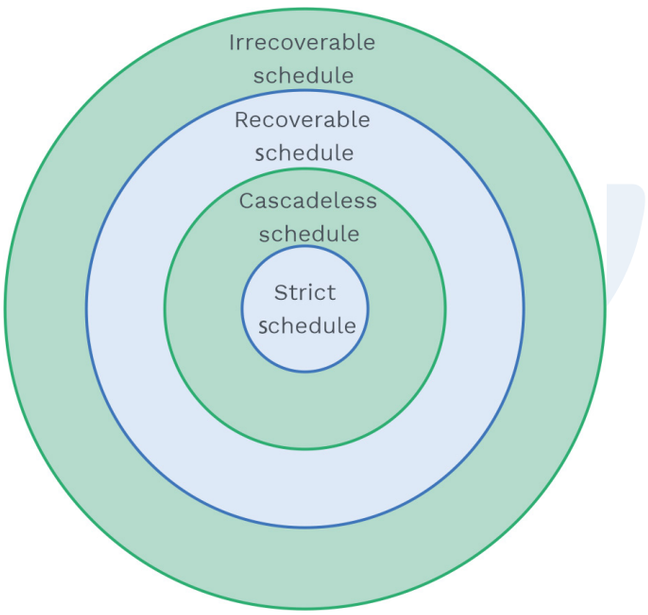
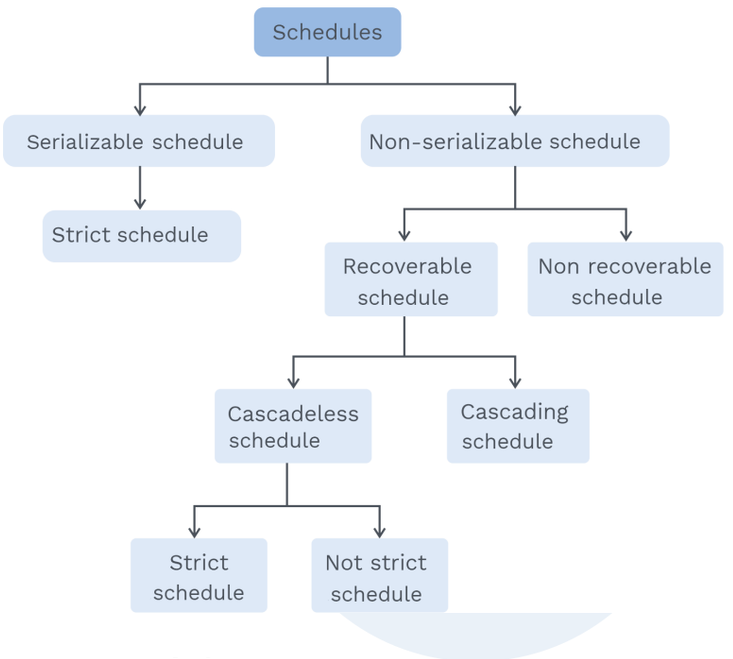
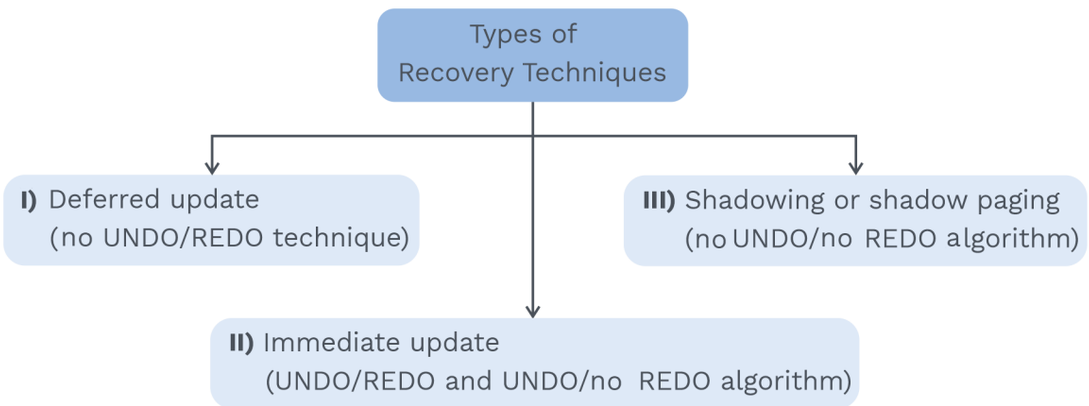


Fig. 4.3 Different Type of Schedules Based on Recoverability



Database recovery techniques

- Failures can occur due to system crashes or transaction errors, etc.
- There are few techniques that we can use to recover from these failures.
- The recovery process uses commit point, system log concepts to recover from any failure.



Checkpointing in the system log

- 'Checkpointing is a type of entry in the log'.
- Generally a record is written into the log periodically and all the updates are recorded on the disk during checkpointing.
- So, transactions that are committed in the log before checkpointing needs not to execute their WRITE operation if any system crash occurs.

- Usually, log records are denoted as:
- <T_i, start> × Transaction T_i has started.
- <T_i × X_j, V_1, V_2> × Transaction T_i has performed a write on data item X_j. Before the write, X_j had value V_1 and after the write X_j will have value V_2.
- <T_i, commit>, Transaction T_i has committed.
- <T_i, abort>, Transaction T_i has aborted.

Why we need checkpoints?

1) It is very time consuming to search the entire log in case of any failure, thus it is always a good idea to maintain checkpoints in the log.

2) Checkpoint also increase the throughput of the system as we need not to waste time in a recovery of a transaction in case if we need to execute that transaction again due to any reason.

UNDO/REDO recovery algorithm with checkpoints

1) There are two list of transactions that needs to be maintained by the system:
   - **i)** The active transactions
   - **ii)** Committed transactions since the last checkpoint.

2) Perform UNDO for every write operation of the uncommitted/active transaction.

3) Perform REDO for every write operation of a committed transaction.

Example: Consider the given checkpointing protocol and the operations given in the log.

(start, T_1)
(write, T_1, y, 3, 2)
(start, T_2)
(commit, T_1)
(write, T_2, z, 4, 7)
(checkpoint)
(start, T_3)
(write, T_3, x, 1, 8)
(commit, T_3)
(start, T_4)
(write, T_4, z, 7, 2)

Now, if crash occurs,the system will try to recover. The undo and redo operations are as follows:
Undo list: T_2, T_4 (It means transaction T_2 and T_4 will be undone.)
Redo list: T_3 (It means transaction T_3 will be redone.)

4.6 CONCURRENCY CONTROL TECHNIQUE

- We know that isolation is one of the important properties that a transaction needs to follow.
- It is mandate while concurrent implementation of transactions to preserve isolation property.
- A concurrency control technique is used to achieve this.

Note:

Concurrency control protocols guarantee serializability.

4.7 LOCK-BASED PROTOCOLS

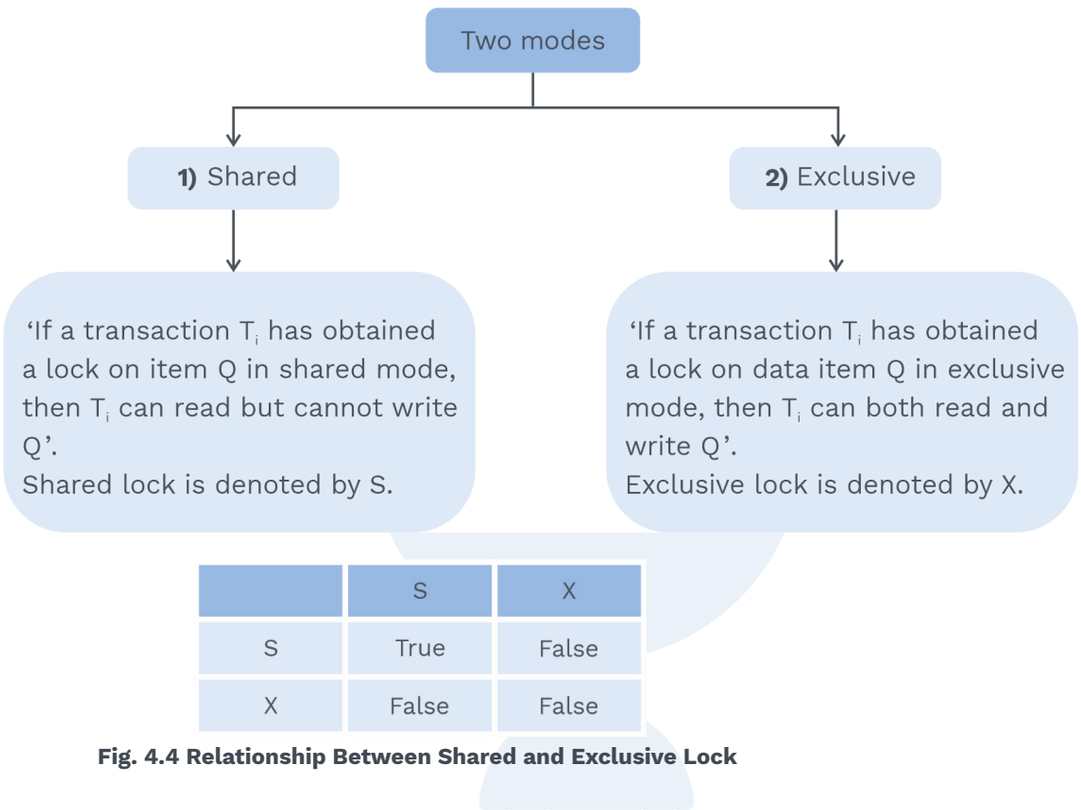
- If we follow mutual exclusion to access the data items, then we can ensure the serializability.
- It means any other transaction cannot modify the data items while the same data item is being accessed by some other transaction.
- We use lock based protocols for this purpose.
- A transaction can have access to a data item if any lock on that specific item is currently held by that transaction.

Definition

'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'.

Lock

- There prevails two distinct locking modes: shared and exclusive.



	S	X
S	True	False
X	False	False

Fig. 4.4 Relationship Between Shared and Exclusive Lock

- A transaction acquires a lock in a particular mode based on the types of operation it wants to perform on the data item Q.
- The concurrency control manager is going to grant these locks to a transaction.
- After getting the locks, a transaction can proceed.

Note:

- Figure 4.4 represents that a shared mode is compatible with shared mode but it is not compatible with exclusive mode.
- The instruction lock-S(A) imparts shared lock on data item, A.
- Similarly, To get a exclusive lock on data item A, we have to use lock-X(A) instruction.
- To unlock a data item A, we have to use unlock(A).
- Locking a data item by a transaction is mandate to have access on it.
- If a transaction has acquired an incompatible lock on a data item, no other lock can be provided to another transaction to access the same data item until that lock has been released. In this case, another transaction has to wait.

Difference between shared and exclusive lock

Shared lock imparts read access on a specific data item. However, exclusive lock mode imparts both read and write grants to a transaction on a targeted data item.

Example: Folks!!

Consider there are two accounts called A and B.
Transactions T_1 and T_2 can access these two accounts.
Suppose transaction T_1 transfers rs. 50 to account A from account B.
Total amount (A+B) is displayed by transaction T_2.

T_1 :	T_2 :
L_X (B);	L_S (A);
R (B);	R (A);
B = B – 50;	U (A);
W (B);	L_S (B);
U (B);	R (B);
L_X (A);	U (B);
R (A);	display (A + B);
A = A + 50;	
W (A);	
U (A);	

Note:

In the above example, L represents lock, R represents read_item, W represents write_item and U represents unlock. We will consider these notation further also.

- Let the value of A and B is ₹ 100 and ₹ 200 initially. If we execute T_1, T_2 serially then T_2 will display A + B = ₹ 300.
- But if these transactions are executed concurrently, then there may be the case possible such that T_2 will display incorrect result.

Example:

T_1	T_2
L_X(B)	
R(B)	
B = B – 50	
W(B)	
U(B)	
	L_S(A)
	read(A)
	U(A)
	L_S(B)
	read(B)
	U(B)
	display (A + B)
L_X(A)	
read (A)	
A = A + 50	
Write (A)	
U(A)	

→ Produces inconsistent result.

- Releasing the lock too early in the simple locking case will lead to inconsistent result.
  (T_2 produces an inconsistent result as T_1 is unlocking the data items too early.)
- Simple locking can also lead to an undesirable situation.

Example:

T_1	T_2
L_X(B)	
read(B)	
B = B – 50	
Write(B)	
	L_S(A)
	read(A)
	L_S(B)
L_X(A)	

Here, the above example is leading to deadlock as T_2 wants a shared lock on B but T_1 holds an exclusive lock on B.

Similarly, T_1 wants an exclusive lock on data item A that is held by T_2 in shared mode. Thus, T_1 is waiting for T_2 to unlock A and T_2 is waiting for T_1 to unlock B.

Grey Matter Alert!

Hey Learners!!
Do you know what happens when a deadlock is present?
When none of the transactions is able to proceed with its normal execution. This situation is known as deadlock.

Note:

If deadlock persists, either of the transactions must undergo roll-back/abort by the system.

Drawbacks of simple locking

i) It leads to an inconsistent result.
ii) Simple locking can also lead to deadlock.
iii) Simple locking does not guarantee serializability.

Example of simple locking schedule that does not guarantee serializability is as follows:

T_1	T_2
lock_X(A)	
write(A)	
Unlock_X(A)	
	lock_S(A)
	read(A)
	Unlock_S(A)
lock_X(A)	
write(A)	
Unlock_X(A)	

This is not conflict serializable schedule as to when we will draw precedence graph, cycle will be present.

Conflict operations are:

1) $W_1(A) \rightarrow R_2(A)$

2) $R_2(A) \rightarrow W_1(A)$

How to grant locks?

- When a transaction asks for a lock on a data item in a particular mode and there is no other transaction that holds a lock on the same data item in a conflicting mode, then a lock can be granted.
- Starvation is a situation where a particular transaction does not make progress as every time a lock is granted to other transaction.

Two phase locking protocol

- Two phase locking protocol always guarantee serializability.
- In 2PL, locking and unlocking is done in 2 phases:
  1) **Growing phase:** 'A transaction can acquire new locks on items but no other lock can be released'. It is also known as expanding phase.
  2) **Shrinking phase:** 'During this phase a transaction can release existing lock but cannot acquire new locks'.

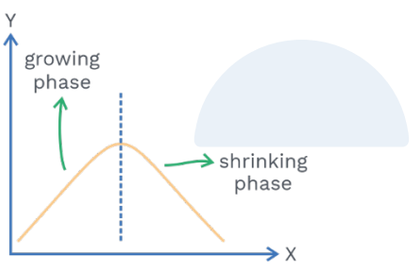


Fig. 4.5 Graphical Representation of 2PL

Note:

Initially, a transaction obtains locks, and it is said to be in growing phase.
After that, a transaction enters the shrinking phase.
Once a transaction initiates lock release phenomenon, it cannot generate any further lock acquisition requests.

T_1	T_2	T_3	
	L-S(A)	L-S(A)	
	L-X(B)		
	U(A)	L-X(C)	
	U(B)		(After taking/obtaining all the locks only, unlocking begins)
L-S(B)		U(A)	
		U(C)	
L-X(A)			
U(A)			
U(B)			

Note:

If two phase locking protocol is followed by all the transactions of a schedule then it is definitely a serializable schedule.

Note:

Same as basic 2PL, 2PL with lock conversion gives only conflict serializable schedule.

Grey Matter Alert!

Lock upgradation	Lock downgradation
To upgrade a shared lock to an exclusive lock.	To downgrade an exclusive lock to a shared lock.

- Upgradation is allowed only in the growing phase whereas down gradation is allowed only in the shrinking phase.
- In the following example, W(A) and W(B) will not be allowed since T_i has exclusive lock.

T_i	T_j
L-X(A) W(A)	
	W(A) W(B) → Blocked
L-X(B) W(B)	

So, how can we determine the order in which serializability is obtained? For that, there is something called a lock point.

Lock point

Definition

Point at which a transaction gets its final lock.
Lock point is used to determine the order of the transactions.

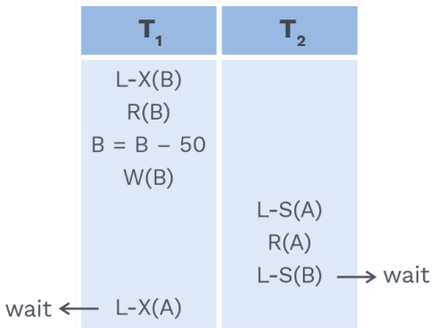
Drawbacks of two-phase locking protocol

1) 2PL may leads to cascading rollback.
2) Deadlock might be possible.
3) Starvation is also possible in 2PL.
Example: Consider a schedule S such that

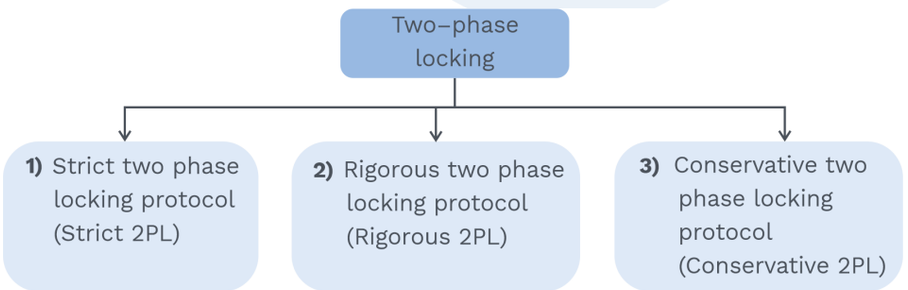
T_1	T_2	
L_X(A)		
W(A)		
L_X(B)		
W(B) ⋮		
U(A)		
U(B)		
	L_S(A)	If T_1 rollback then T_2 also need to rollback
	R(A)	

Rollback ↑

- Example detecting deadlock.



- Here, T_1 is having exclusive lock on data item B and is requesting exclusive lock on data item A.
- Similarly, T_2 is having shared lock on data item A and requesting shared lock on data item B, which clearly says that deadlock is present.
- Which clearly says that deadlock is present.
- There are three types of two-phase locking.



Strict 2PL

- Strict 2PL guarantees strict schedules.

Definition

'It is a variation of 2PL, a transaction T does not release any of its exclusive (read and write) locks until it commits or aborts'.

- So, any other transaction cannot read or write an item written by T until T has committed.
- It leads to a recoverable schedule.
- Strict 2PL ensure that a schedule is:
  1) Recoverable schedule **2)** Cascadeless schedule

- The advantage of strict 2PL is that it avoids cascading rollbacks.
- Strict 2PL is not deadlock free.

Example:

T_1	T_2
L-X(A) W(A) Commit U(A)	
	L-S(A) R(A) U(A) Commit

(Since lock on A is already given to T_1, it will never be granted to T_2, that is T_2 will not able to read/write same data item until transaction T_1 that has written the data item performs commit.)

Note:

Hey Learners!!
Do you know how can we decide the order of transactions in the strict 2PL?
Well, the order of transactions is decided by the sequence of lock points.

Whatever final order, we get is equivalent to serial schedule.

Rack Your Brain

Strict 2PL protocol guarantees:

1) Recoverable schedule
2) Cascadeless schedule
3) Strict schedule
4) All of the above

Previous Years' Question

Consider the following database schedule with two transactions T_1 and T_2.
$S = r_2(x); r_1(x); r_2(y); w_1(x); r_1(y); w_2(x); a_1 ; a_2$
Where $r_i(z)$ denotes a read operation by T_i on a variable z and a_i denotes an abort by transaction T_i.
Which one of the following statements about the above schedule is TRUE?

1) S is non-recoverable
2) S is recoverable, but has a cascading abort
3) S does not have a cascading abort
4) S is strict

Sol: 3). **(GATE-2016, Set-2)**

Rigorous 2PL

- Rigorous 2PL is more restrictive variation of strict 2PL.

Definition

'In rigorous 2PL, in addition to locking being in 2 phase, a transaction T does not release any of its locks (shared or exclusive) until it commits or aborts'.

- Rigorous 2PL also guarantees strict schedules.

Conservative 2PL

Definition

In conservative 2PL, all the locks acquired will not be released until transaction commits.

- It says that once the transaction begins it is in its shrinking phase and then transaction is in its expanding phase until it finishes.

- Advantages of conservative 2PL:
  1) Conservative 2PL gives strict schedules.
  2) It also gives freedom from deadlock.

Note:

Conservative 2PL may lead to starvation.

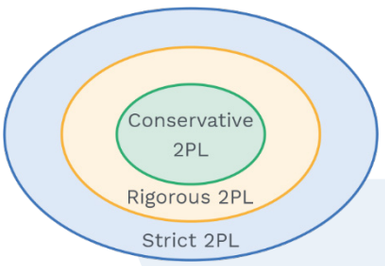


Fig. 4.6 Relation Between Strict, Rigorous and Conservative 2PL

Deadlock

Definition

'Deadlock is said to occur when each transaction T in a set of two or more transactions is waiting for some item that is locked by some another transaction T' in the set'.

4.8 MULTIPLE GRANULARITY

- It is better if we can group several data items and consider them as an individual units instead of considering each individual data item on which we will perform synchronisation.

Example: Let's make an assumption that a transaction T_1 demands access over the whole database utilising the concept of locking protocol. Each item in the database must be locked by T_i. This process is time consuming. In place of this, we can allow T_i to issue a single lock acquisition plea. Differently, if transaction T_j wants to access only few data items, then no need to lock the entire database.

- Hence, there exists scope to establish the idea of multiple granularity.
- We can achieve this by allowing various size data items and by defining a hierarchy of data granularities.
- We can represent this hierarchy as a tree.
- A non-leaf node represents the data associated with its descendants.

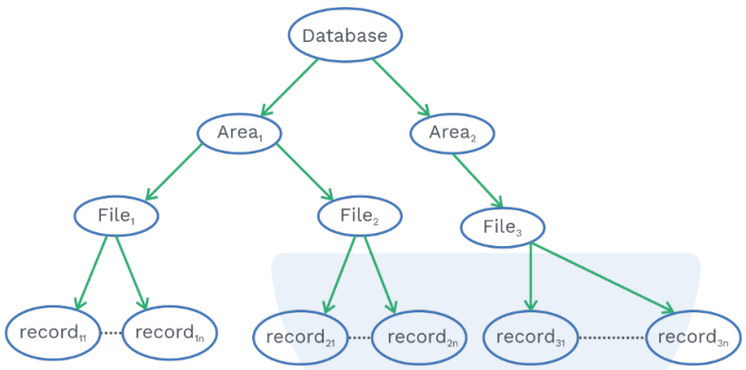


Fig. 4.7 Granularity Hierarchy

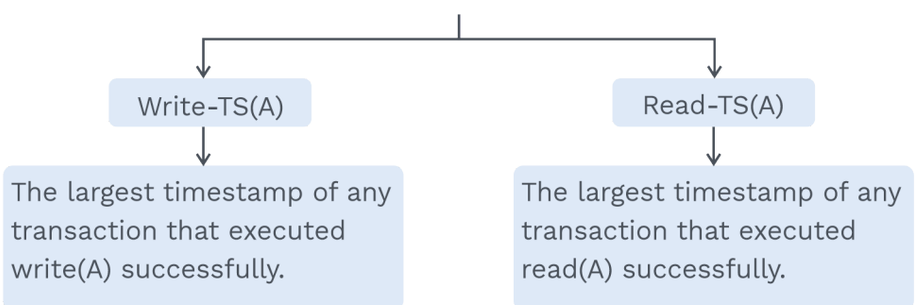
4.9 TIMESTAMP BASED PROTOCOLS

Note:

If we provide a timestamp to each transaction, then we can prevent deadlock.

Definition: 'Transaction's Timestamp is a unique identifier assigned to each transaction'.

- We assign timestamp to the transaction based on when a transaction has arrived.
  TS (T_1) < TS (T_2) means transaction T2 comes after T1.
- The serializabiltiy order can be found using the transaction timestamp.
- We associate 2 timestamp values with each data item A.



- Timestamps will get updated if new write or read instruction is executed.

Timestamp ordering protocol

- We know that timestamp ordering based concurrency control techniques do not use locks.
  Therefore no deadlock will occur.
- In this protocol any conflicting write or read operation will execute in timestamp order.
- If it is not then such an operation is rejected and the transaction will be rolled back.

Note:

Whenever the concurrency control scheme rolls back a transaction, the system provides it with a new timestamp and restarts it.

Example: Consider a schedule:

T_1	T_2
R(A)	
	W(A)
W(A)	

Suppose TS (T_1) = 1 and TS (T_2) = 2 then conflicting action $R_1(A) \rightarrow W_2(A)$ is allowed as TS (T_1) < TS (T_2) but conflicting action $W_2(A) \rightarrow W_1(A)$ is not allowed as transaction having greater timestamp has already executed. So, it will be rolled back and restarted again with different timestamp.

Example: Consider an example:

Suppose TS (T_1) = 1 and TS (T_2) = 2 and following schedule is given.

T_1	T_2
R(A)	
	R(A)
W(A)	

In this example there is no conflict operation from transaction T_1 to T_2 but there is a conflict operation from transaction T_2 to T_1. Since TS (T_1) < TS (T_2) so, conflict operation from T_2 to T_1, i.e. $R_2(A) \rightarrow W_1(A)$ is not allowed. Thus, this transaction T_1 will rollback and restart again with a new timestamp. Let us say TS (T_2) = 2 and TS (T_1) = 3.

T_2	T_1
R(A)	
	R(A)
	W(A)

T_2T_1 is equivalent serial schedule.

Note:

In basic timestamp order protocol, if there are two conflict operations that occur in the incorrect order, then we can reject later of the two atomic operation through abort of the targeted transaction.
So, conflict serializability is secured through timestamp ordering.

Strict timestamp ordering

- A strict timestamp ordering ensures strict and serializable schedule.

Definition

- 'In strict timestamp ordering, a transaction T that issues a read_item (A) or write_item (A) such that TS (T) > write_TS (A) has read or write operation delayed until the transaction T' that wrote the value of A has committed or aborted'.
- The strict timestamp ordering avoids deadlock.

4.10 DEADLOCK HANDLING

There are different ways to handle a deadlock, for example, deadlock prevention, deadlock detection and recovery.

Deadlock prevention

Note:

There prevails two mechanisms to ascertain deadlock prevention: wait-die, wound-wait.

Definition

Time-stamp: It is represented as TS(T_i) and used to prevent deadlock. Transaction timestamp is a unique identifier that is assigned to each transaction.

1) Wait-die

- If TS (T_m) < TS (T_n), means T_m is older than T_n.

Case 1: When an older (T_m) transaction tries to lock an element that is already locked by younger (T_n) transaction then the older (T_m) transaction has to wait.
Otherwise,
Case 2: When younger (T_m) transaction tries to lock an element that is already locked by older (T_n) transaction then the younger (T_m) transaction dies.

2) Wound wait

- If TS (T_i) < TS (T_j), means T_i is older than T_j.

Case 1: When an older transaction (T_i) tries to lock an element that is already locked by younger transaction (T_j) then T_i wounds T_j.
Otherwise,
Case 2: When a younger transaction (T_i) tries to lock an element which is already locked by older transaction (T_j) then T_i has to wait.

- In both methods of preventing deadlock, younger of the two transactions where the deadlock is present, get aborted.
- Both techniques are deadlock free as no cycle is possible in any of these techniques.

Deadlock detection and recovery (wait for graph)

Definition

'In deadlock detection, system checks whether the deadlock exists or not'.

- It helps in the detection of deadlock existence.
- A directed edge will be formed if transaction T_1 is waiting to lock an item that is currently locked by transaction T_2.
- Remove the directed edges from the graph if the lock is released by T_j on those items for which T_i was waiting.
- Deadlock is present in the wait for graph if and only if graph has a cycle.

Example: Consider the figure given below:

$T_1 \xrightarrow{\text{Waiting for}} T_2, T_3$

$T_3 \xrightarrow{\text{Waiting for}} T_2$

$T_2 \xrightarrow{\text{Waiting for}} T_4$

$T_4 \xrightarrow{\text{Waiting for}} T_3$

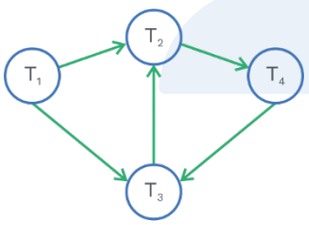


Diagram representing wait for graph with a cycle.

Graph-based protocols

- There is a graph-based protocol that does not use 2PL.

It is also called as tree protocol.

Note:

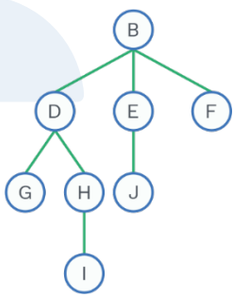
The main advantage of the tree protocol is that we can completely avoid deadlocks.

- We only use exclusive locks in the tree protocol.
- Any data item can be locked at most once by each transaction T_i.
- These are the following rules:

1) Any transaction, T_i is permitted to submit lock acquisition request on any data item.
2) T_i is permitted to lock data item, X if it has lock grant over parent of X.
3) Lock release can occur at any time instant.
4) The transaction T_i cannot be granted lock on the same data item after unlocking it previously.

Example: Consider the given below schedule:

T_1	T_2	T_3	T_4
lock-X(B)			
	lock-X(D)		
	lock-X(H)		
	unlock-X(D)		
lock-X(E)			
lock-X(D)			
unlock (B)			
unlock (E)			
		lock-X(B)	
		lock-X(E)	
	unlock (H)		
lock-X(G)			
unlock (D)			
			lock-X(D)
			lock-X(H)
			unlock (D)
			unlock (H)
		unlock (E)	
		unlock (B)	
unlock (G)			



Thomas write rule

Thomas write rule rejects fewer write operations by modifying the checks for the write_item (Q) operation as follows:

1)	If read_TS (Q) > TS (T), then abort and roll back T and reject the operation.
2)	If write_TS (Q) > TS (T), then do not execute the write operation but continue processing.
3)	If none of the above two condition occurs, then execute the write_ item (Q) operation of T and set write_TS (Q) to TS (T).

Example: Schedule are given below:

T_1	T_2
R(A)	
	W(A)
W(A)	

Here, in Thomas write rule, $W_2(A) \rightarrow W_1(A)$ is allowed, no rollback.

Note:

T_1 and T_2 are transactions such that time stamp of T_1 < time stamp of T_2. Then:

	Not Allowed	Allowed
Basic time stamp ordering protocol (similar to conflict serializable)	$R_2(A) \rightarrow W_1(A)$ $W_2(A) \rightarrow R_1(A)$ $W_2(A) \rightarrow W_1(A)$	$R_2(A) \rightarrow R_1(A)$
Thomas write time stamp ordering protocol (similar to view serializable)	$R_2(A) \rightarrow W_1(A)$ $W_2(A) \rightarrow R_1(A)$	$R_2(A) \rightarrow R_1(A)$ $W_2(A) \rightarrow W_1(A)$

Chapter Summary

- **Transaction:** A collection of operations that forms a single logical unit of work.
- **ACID properties.**

  There are 4 ACID properties that a transaction needs to follow:

1)	A	⟶	Atomicity
2)	C	⟶	Consistency
3)	I	⟶	Isolation
4)	D	⟶	Durability

- **Types of failure in a system:** Transaction failure, system crash, disk failure, power failure, software crash, natural hazards, etc.
- **Transaction states:** There are five states for a transaction namely active, partially committed, failed, committed, aborted through which a transaction goes in its lifetime.
- Transaction processing system allows multiple transaction to run concurrently.
- **Concurrency problems in transactions:**
  1) Reading uncommitted data (W-R)
  2) Unrepeatable read (R-W)
  3) Overwriting uncommitted data (W-W)
  4) Phantom read problem
- **Serial schedule:** Serial schedule are those schedule where operations of each transaction executes consecutively.
- **Complete schedule:** When the last operation of each transactions is commit or abort the operation. That schedule is known as complete schedule.
- **Serializability:** Serializability is a property that indicates the correctness of the schedule when a concurrent transactions are executing.
- **Types of schedule based on serializability:**
  1) Conflict serializable schedule.
  2) View serializable schedule.
- **Types of schedule based on recoverability:**
  1) Irrecoverable schedule
  2) Recoverable schedule

3) Cascade rollback recoverable schedule
4) Strict recoverable schedule

- Recoverable schedules ensures that, once a transaction commits, it never rollbacks.
- If one transaction failure causes multiple transactions to rollback, it is called as cascading rollback.
- A serializability order of the transactions can be obtained through topological sorting of precedence graph.
- There are different type of concurrency control techniques such as lock-based protocols, timestamp based protocol.
- **Deadlock:** When none of the transactions is able to proceed with its normal execution, this situation is known as deadlock.
- **Two techniques to handle deadlock:**
  1) Deadlock prevention (wait-die and wound wait)
  2) Deadlock detection and recovery
- To prevent deadlock, there is a concept called as transaction timestamp denoted as TS (T_i) which is a unique identifier assigned to each transaction.

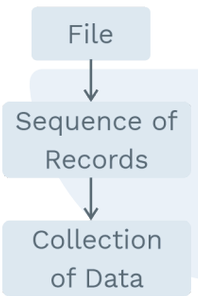
5 File Organization & Indexing

5.1 FILE ORGANIZATION AND INDEXING

Introduction:

- In a system, a database is stored in the form of files containing records.
- These files of records are stored in secondary memory like magnetic disks.
- Each file is partitioned into fixed-length blocks.
- One block can contain more than one data item.
- One of the reasons to use a database system is to decrease the number of blocks that are transferred between disk and memory.

File organization:



- A file is organized as a sequence of blocks.
- A block is organized as a sequence of records.
- Records means a collection of data items(fact).

Primary file organization:

- It describes how can we place file records on the disk as well as how can we access them based on the primary field.

Secondary file organization:

- In this, we can access the records of files based on alternate fields (not primary fields).

Note:

File organization refers to the organization of the data of a file into records, blocks and access structure.

- As we know, a block is the smallest unit that is used to transfer the data between disk and memory.

- So, we need to allocate records of the file to disk blocks.
- When block size > record size, then one block can have many records.
- But, we can not fit these records to one block if the number of records present are large.

Grey Matter Alert!

- The file is termed as fixed-length records where all the records are of exact same length.

Blocking factor:
It is defined as the maximum number of records that can be stored in a disk block.
Let us consider, BS > RS where,
BS represents block size in bytes and
RS represents size of the fixed length records.
Average number of records per block is known as Blocking factor.
We can define blocking factor = $\lfloor BS/RS \rfloor$ records per block. $\lfloor\ \rfloor$ represents floor function.

Note:

Sometimes record size does not divide block size exactly.
So, there will be some unused space = (BS- (Blocking factor * RS)) bytes.

- Records can be organized in a database file using the following two strategies:
  1) Spanned
  2) Unspanned

1) Spanned: In this strategy, records are allowed to span in more than one block. i.e. it allows partial part of records to be stored in a block.
Example: Consider the figure given below, which is depicts the spanned strategy for storing files of records into a block.

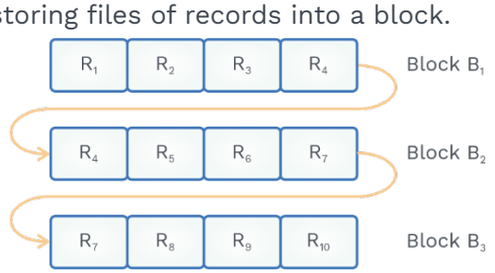


Fig. 5.1

In this example, record R_4 is stored in block B_1 as well as Block B_2. Similarly record R_7 is stored in block B_2 as well as B_3.

Advantage of spanned strategy:
No wastage of memory.

Disadvantage of spanned strategy:
Number of block accesses to access a record increases.

Note:

This strategy is appropriate for storing those records whose length is variable.

2) **Unspanned strategy:**
The records are not allowed to cross block boundaries, i.e. if no record can be stored in more than one block then this strategy is known as unspanned strategy.
Example: Consider the following example given below that depicts unspanned strategy for storing records.

Fig. 5.2

Advantage of unspanned strategy:
Number of block accesses to access a record reduces.

Disadvantage of unspanned strategy:
Wastage of memory.

Note:

This strategy is appropriate for storing those records whose length is fixed.

Note:

If the average record size is large, it is beneficial to use spanned organization, to reduce the space that is lost in each block.

- The Number of blocks b required for a file of a 'r' records

$$b = \lceil (r/\text{blocking factor}) \rceil \text{ blocks}.$$

Where ⌈ ⌉ is a ceiling function and it rounds the value up to the next integer.

SOLVED EXAMPLES

Q1 **Consider a file of size 2 MB, having 2K fixed length Blocks. Following are the records which need to be stored in the blocks: R1(500B), R2(800B), R3(300B) R4(1000B), R5(500B). If spanned organization of records is used. What are the total number of Blocks required to store all the records?**
1) 5 blocks 2) 4 blocks 3) 6 blocks 4) 3 blocks

Sol: **2)**

In spanned organization records are allowed to span in more than one Block.

$$\text{Block size} \frac{\text{File size}}{\text{Number of Blocks in File}}$$

$$= \frac{2 \times 2^{20}\text{B}}{2 \times 2^{10}} = 1\text{KB}$$

500+800+300+1000+500 = 3100 bytes
So, number of blocks required = ⌈3100 bytes/1024 bytes⌉
= ⌈ 3.02 ⌉
= 4
Hence, 4 Blocks are required to store all the records.

Allocation of file blocks on disk:

- File blocks can be allocated on the disk using the following techniques:
- **Continuous allocation:** In continuous allocation, the file blocks are allocated to consecutive disk blocks.
- **Linked allocation:** In linked allocation; one file block is linked to the other(next) file block with a pointer.
- **Disadvantage:** This allocation makes it difficult and slow to read the whole file.
- **Indexed allocation:** In Indexed allocation, one or more index blocks contain pointers that point to the existing file-blocks.

Files of unordered records:

- All records in file are inserted wherever there is place available for the record.
- There is no ordering of records.
- Files of unordered records are also known as Heap files.
- Any particular record is searched using linear search.
- **Advantage:** Inserting a record is efficient.
- **Disadvantage:** Searching for a record is inefficient; it is expensive procedure due to the involvement of linear search.

Note:

- For a file having 'b' blocks, on average, this requires searching (b/2) blocks.

Files of ordered records:

- **Advantage:**
  1) As there is no sorting required, we can efficiently read the records.
  2) There is no requirement for additional block access.
  3) Binary search is used to search for a record in a file that consists of records in an ordered way, so access of to records will be faster.

Note:

A binary search access $\log_2(b)$ blocks whether the record is found or not.

Average block access required for a file having 'b' blocks in basic file organization

	Types of Organization	Access/Search Method	Average Blocks To Access A Specific Record
1)	Heap (unordered)	Linear Search (Sequential scan)	b/2
2)	Ordered	Linear Search (Sequential scan)	b/2
3)	Ordered	Binary Search	$\log_2 b$

Table 5.1

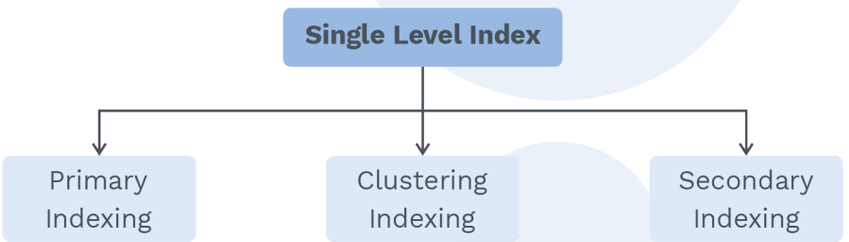
Index structure:

- Indexes are the auxiliary access structure.
- It is used to increase search efficiency.
- Using the index structure, we can easily access the records without creating any disturbance to the records that are placed physically in the main data file on disk.
- We can access records efficiently based on the indexing fields.
- Following are the different types of indexes used
  1) Single-level index
  2) Tree data structures
  (multilevel index, B+ trees).

Grey Matter Alert!

- There is another type of primary organization which is based on hashing.
- It provides very fast access to records under certain conditions.
- This organization is called the hash file.

5.2 SINGLE LEVEL INDEX



- An index structure is based on a field termed as indexing field.
  1) Primary index:
- It is an ordered-file where length of the records are fixed and contains 2 fields.
- The 1st field consists the data type, which is same as the primary key of the file where data is present.
- The second field consists a pointer pointing to a disk block.
- For every block present in the data file, there must be an index record in index file.

Definitions

"A primary index is specified on the ordered key field of an ordered file of records."

Note:

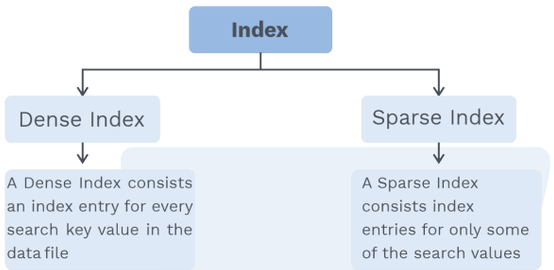
A Number of index-entries =Number of disk blocks.

- The average number of block accesses using primary index = ⌈log2 Bi⌉ + 1. where Bi refers to the total block count.

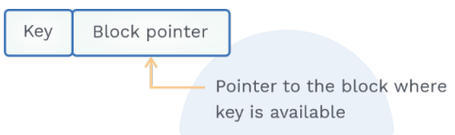
Note:

- Hey learners!!
  Do you know what an anchor record is?
  An anchor record is the first record in each block of the data file.
- Anchor record is also called as block anchor.

- Indexes are also characterized into two categories dense and sparse index.



- Structure of index files:
  Index record consists of two fields: key and block pointer.



- A sparse index stores lesser count of entries in comparison to the count of records in the original database file.

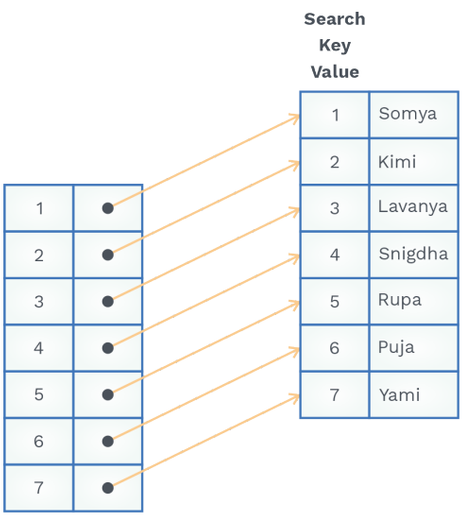


Fig. 5.3 Dense Index

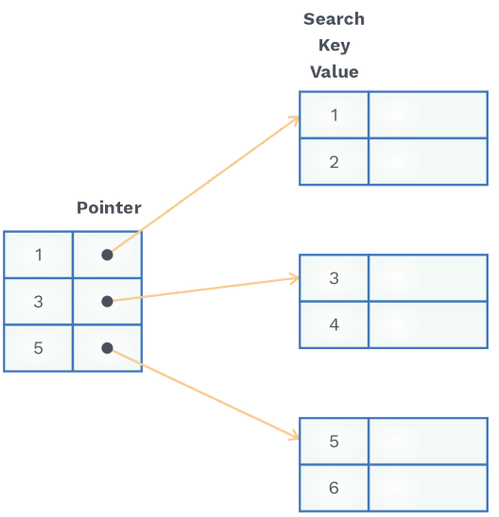


Fig. 5.4 Spare Index

- As we can see in Fig. 5.1, for every search key, there exists an entry in the index file. Thus it is classified as dense index.
- But, in Fig. 5.2, index entries are not available for every search key value. For every disk block, there is subsists an index entry, thus, it is classified as sparse Index.

SOLVED EXAMPLES

Q2 **Consider an ordered file that contains 30,000 records. These records are stored in a disk. The Size of a block is 1024 bytes. The length of file records are is 100 bytes and records cannot be spanned. A primary index structure is employed that contains 9 bytes key and 6 bytes block pointer. Estimate the mean number of block access needed without indexing and with indexing?**

Sol: **Given, size of the block = 1024 bytes**

Size of each record = 100 bytes

Number of records per block $= \frac{1024}{100} = 10.24$

Since records are stored in unspanned manner, we can store only 10 records at maximum.

Number of blocks required to store 30000 records $= \frac{30000}{10} = 3000$

Without primary index

Number of block accesses $= \lceil \log_2 3000 \rceil = 12$

Size of a index record = 9 bytes + 6 bytes = 15 bytes

Number of index record per block $= \frac{1024}{15} = 68$

As we got the number of blocks that is are required to store all these records are= 3000

So, the total number of index records = 3000

$\because$ 68 index records are present in 1 block.

3000 index records will be present in 3000/68 = 45 block.

With primary index

Total number of block accesses $= \lceil \log_2 45 \rceil + 1$

$= 6 + 1 = 7$

Drawbacks of primary indexes:

- The main drawback of a primary index is insertion of records and deletion of records.
- Movement of records is needed to put the new records in the right place.
- But, this will also lead to changes of some of in the index entries.
- **Reason:** Due to the movement of records, it is a high possibility that anchor-records of some of the blocks might get changed.

Definitions

- "Clustering index is created on data file whose file records are physically ordered on a non key field which may not be unique for every record."

2) Clustering indexes:

- A clustering index is an ordered file with two fields:
  1st field contains the data type that is similar to the clustering field of actual file where data is present.
  2nd field contains a block-pointer.

Note:

Clustering Index: An example of a non-dense index.
In clustering indexing, there is an a unique value for every record in the file.

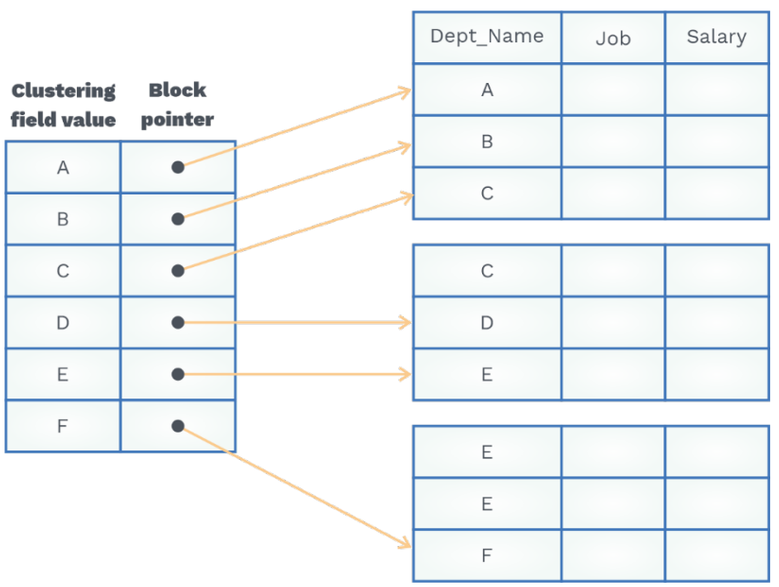


Fig. 5.5 Diagram Represents Clustering Indexing

- The number of block access using clustering index ≥ $\lceil \log_2 Bi \rceil + 1$. where Bi refers to the total block count.

3) Secondary indexes:

Definitions

- A secondary index gives a secondary way to access a data file where primary index or clustering index are already defined.
- Data file records might be ordered, hashed, or unordered.

Note:

The secondary index is based on an unordered field that is
1) CK(Candidate key).
(OR)
2) Non-key field with duplicate values.

The secondary index: ordered file having two fields:
The data type of the first field is same as that of the clustering field of the actual database file.
The second field captures the block pointer.

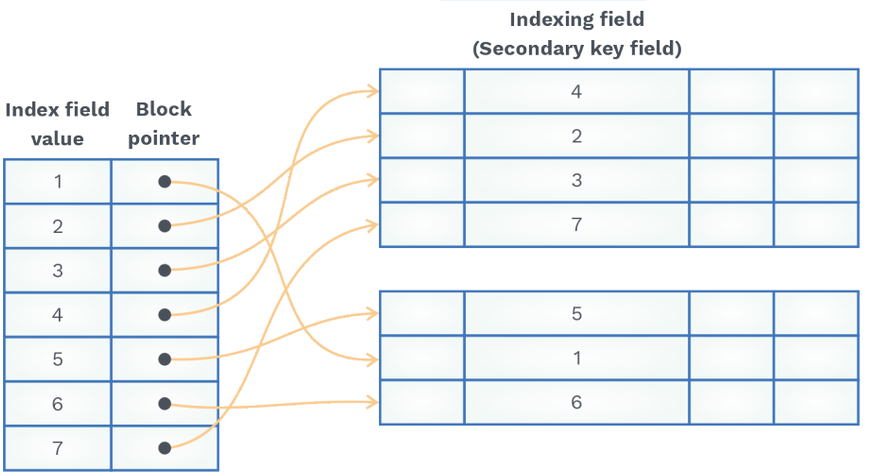


Fig. 5.6 A Dense Secondary Index on a Non-Ordering Key Field of a File

Note:

Number of index entries at first level in secondary index = number of records in the database file.

Previous Years' Question

A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called.

1) Dense
2) Sparse
3) Clustered
4) Unclustered

Sol: Option 3) **(GATE-2015 Set-1)**

SOLVED EXAMPLES

Q3 **Assume a database file with 45,000 tuples. This file is organized block-wise in hard-disk. A secondary index structure is used to access the file with key size of 7 bytes. If the sizes of the block, block pointer and record are 2048, 7 and 100 bytes, respectively. If the database follows an unspanned organization, estimate the feasible count of block accesses with and without utilizing index structure.**

Sol: **Block size = 1024 bytes**

Number of data records per block $= \frac{2048\ B}{100\ B} = 20.48$

We can put a maximum of 20 records only in 1 block.

Number of blocks required by data file $= \frac{45000}{20} = 2250$ blocks

Number of index records = number of data records = 45000
Size of index record = 7 + 7 = 14 bytes.

Number of index records per block $= \left\lfloor \frac{2048}{14} \right\rfloor = 146$ entries per block

45,000 entries with 146 entries/block: $= \frac{45,000}{146} = 308$ blocks for the index.

So, without using secondary indexing, we will perform linear search on the file that is equal to $\frac{b}{2} = \frac{2250}{2} = 1125$ block accesses on the average.

With indexing:
A binary search on this secondary index needs $\lceil \log_2 308 \rceil$ = 6 block accesses.

We need one more block-access for searching a record if secondary indexing is used.
Therefore, Total block-access=
6 block access + 1 extra block access = 7 block accesses.

5.3 MULTILEVEL INDEX

- A binary search takes log2bi block access for an index having bi blocks.
  Reason: In each step, part of the index file will reduce by a factor of 2.
- That is why we use a log to the base 2 as a function.
- Due to multilevel indexing, searching becomes much faster.
  Do you know why we need multilevel indexing?
- Sometimes, we have a larger database, and we cannot fit indexes in a single block of a disk.
- Therefore, we need multilevel indexing.
- In multilevel indexing, in order to access a disk block, firstly outer level
- index block is accessed, and a particular entry in the outer level index points to a block in the inner level index block which will point to the disk block which containing the required record.
- y I/O cost to access a record using multilevel indexing having K levels = K+1.

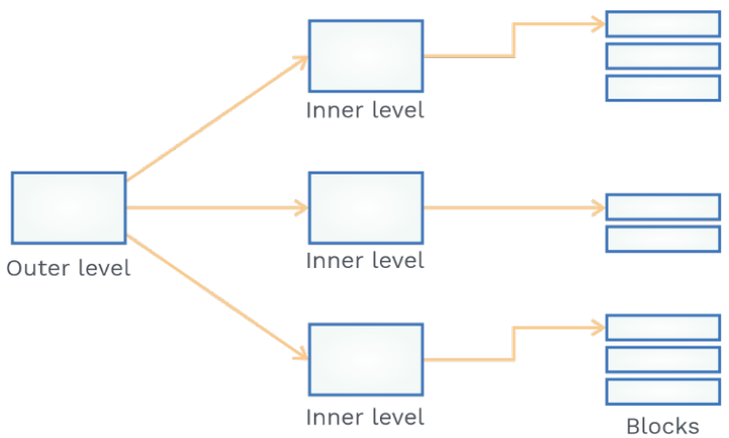


Fig. 5.7

B Tree and B⁺ tree:

- B tree and B⁺ trees are special case of well-known search data structure trees.

Search trees:

Definitions

A search tree is used to search if a record exists or not in the relation based on a given set of field values.

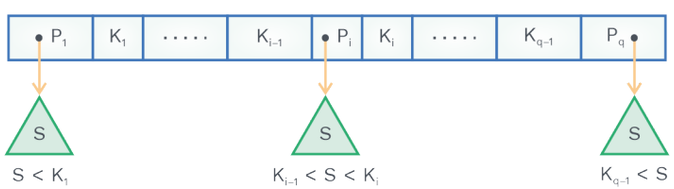
- Suppose P is the order of the Search tree. Then:
- **1)** Maximum search key a node can contain=(P-1)
- **2)** Maximum block pointers a node can contain=P
- **Structure:** $<P_1, K_1, P_2, K_2, ..., P_{m-1}, K_{m-1}, P_m>$ where $m \leq P$ (order of tree).
- Where, P_i is a pointer pointing to a child node and Ki is a search key value.
- **Application:** The Search tree is useful for searching records that are present within a disk-file.
- There are two constraints that need to be followed by a search tree.
  - **i)** Every node should contain the search key value such that, $K_1 < K_2 < ... < K_{m-1}$.
  - **ii)** For all values S in the subtree pointed by P_i,

Grey Matter Alert!

Do you know anything about the TREE data structure?

1) "A tree is defined recursively as a collection of nodes."
2) A root node is present in a tree.
3) Except root node, other nodes are having 1 parent.
4) Also, other nodes can have either no child or more than 0 children. Leaf nodes: It is a node with no children.
5) Internal Node : all nodes other than leaf nodes.

We have:



Grey Matter Alert!

A Tree contains the values from one of the file fields, known as search field.

Example:

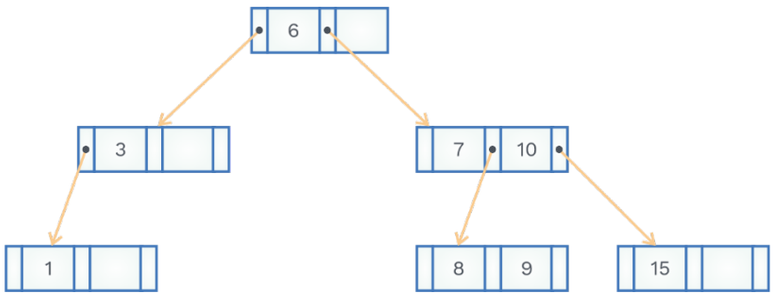


Fig. 5.8 Search Tree of Order P = 3

5.4 B-TREES

- There is an auxiliary condition in B-tree which confirms that the tree is always balanced.
- In B-trees, at every level, we are going to have key and data pointer pointing to either block or record.
- A B-tree of order P can be defined as

  i) Internal node of the B tree is as follows:

$$<p_1, <k_1, pr_1>, p_2, <k_2, pr_2>, \ldots <k_{m-1}, pr_{m-1}>, p_m>$$

where $m \leq p$ (order of tree).

1) Here, each p_i is a tree pointer, and each pr_i is a data pointer.

Tree pointer (Block pointer): A pointer to another node in the B-tree.

Data pointer: It is a pointer that points to the record containing Ki as the search key value.

2) For each internal node, $k_1 < k_2 < k_3 \ldots < k_{m-1}$

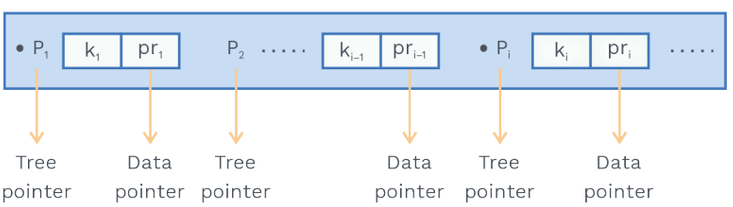
3) Presume S is a search key value in the subtree pointed by p_i. Then:

$K_{i-1} < X < k_i$ for $1 < i < m$

$X < k_i$ for $i = 1$

$K_{i-1} < X$ for $i = m$

4) Maximum count of tree pointers possible is P for each node.



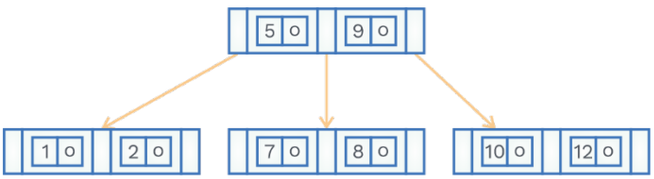


Fig. 5.9 A B-Tree of Prder = 3

B-tree properties:

1) Root node: A root node can have minimum 2 children (Block pointer) and maximum P children.

Where P → order of tree.

Note:

Order of tree: The Order of a tree represents the maximum number of block pointers a node can have. (default definition)

2) Internal nodes:

The Internal node has atleast $\lceil (P/2) \rceil$ tree pointers (Block pointer). It means the internal node can have children between $\lceil P/2 \rceil$ and p.

3) Leaf nodes:

- All leaf nodes are at the same level.
- Also leaf node have the same structure as internal nodes except that all of their pointers p_i are NULL.

Note:

A node with m tree pointers, where m ≤ p(order of the tree) has m–1 data (record) pointers as well as m-1 Search key items(values).

Example: Consider a B-tree of order p = 5. Then the minimum and the maximum number of keys for root, internal and leaf nodes are as follows:

Node	Minimum number of keys	Maximum number of keys
Root	1	4
Internal node	$\lceil 5/2 \rceil - 1 = 2$	p – 1 = 4
Leaf node	$\lceil 5/2 \rceil - 1 = 2$	p – 1 = 4

SOLVED EXAMPLES

Q4 **Consider a B-tree with given data: Size of search key value = 10 bytes, block size of 512 bytes, block pointer is of 5 bytes, and the data pointer is of 8 bytes. What will be the order of the B-tree?**

Sol:

Let n is the order of the B-tree
k represents the key size
p_r represents record pointer
p_b represents block pointer
BS represents block size
A/Q, Given: k = 10 bytes, BS = 512 bytes, p_r = 8 bytes, p_b = 5 bytes.

$n \times p_b + (n - 1) \times (k + p_r) \leq$ Block size

$\Rightarrow \quad n \times 5 + (n - 1) \times (10 + 8) \leq 512$

$\Rightarrow \quad 5n + 18n - 18 \leq 512$

$\Rightarrow \quad 23n - 18 \leq 512$

$\Rightarrow \quad n \leq \frac{530}{23}$

$n \leq 23.04$

$\therefore \quad n = 23$

Thus, the order of B-tree = n = 23.

Q5 **What will be the minimum and maximum number of keys in the B-tree of order 23 for internal nodes?**

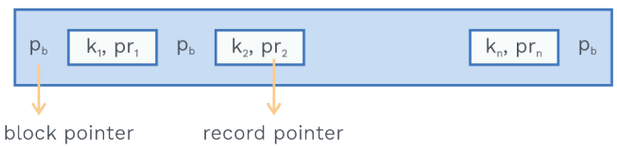
Sol: **For internal nodes,**

Minimum number of keys = $\lceil p/2 \rceil - 1 = \lceil 23/2 \rceil - 1 = 11$

Maximum number of keys = p – 1 = 23 – 1 = 22

Q6 **Consider a B-tree with a search key size of 9 bytes, block size of 512 bytes, record pointer is of 7 bytes, and block pointer is 6 bytes. What is the order of B-tree?**

Sol:



Given, k = key size = 9 bytes, p_r = record pointer = 7 bytes
p_b = block pointer = 6 bytes, BS = block size = 512 bytes
Now, Let 'n' is the order of B-tree

$n(p_b) + (n - 1)(k + p_r) \leq 512$

$\Rightarrow \quad n(6) + (n - 1)(9 + 7) \leq 512$

$\Rightarrow \quad 6n + 16n - 16 \leq 512$

$\Rightarrow \quad 22n \leq 512 + 16$

$\Rightarrow \quad n \leq \frac{528}{22}$

$n \leq 24$

$\therefore \quad n$ = order of B-tree = 24.

Q7 **Suppose order of B-tree is 23. Then how many maximum index records are stored in 4 levels [including root as 1 level] across the B-tree?**

Sol: **Given, The order of the B-tree is 23.**

In level 1: There will be 23 pointers which are equal to the number of block pointer.

Level 1 contains a maximum of 22 index records.

In level 2: In level 1, there are 23 children, and each contains 22 index records at maximum. Thus level 2 will contain at maximum (23) (22) index records.

Similarly, level 3: (23) (23) (22) index records at maximum

level 4: (23) (23) (23) (22) index records at maximum

Thus, overall the maximum index records stored in 4 levels (including root as 1-level) in given B-tree of order 23 = 22 + 23 × 22 + 23 × 23 × 22 + 23 × 23 × 23 × 22

= 22 (1 + 23 + 23 × 23 + 23 × 23 × 23)

= 22 (1 + 23 (1 + 23 + 23 × 23))

= 22 (1 + 23 (1 + 23 (1 + 23)))

= 22 (1 + 23 (553))

= 279840

Searching algorithm in B-tree:

Searching a B-tree is similar to a binary search tree, However rather than moving left or right at each node, we need to perform a p-way search to see which subtree to probe.

Consider an B-tree of order-4.

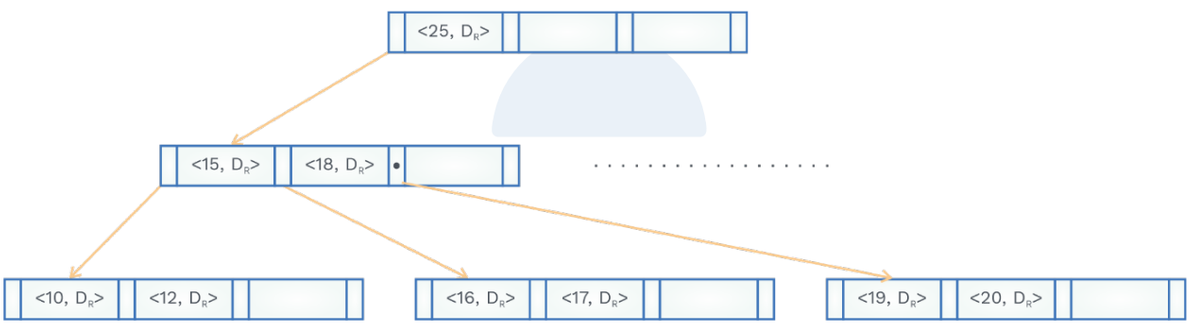


Fig. 5.10 Partially Drawn B-tree of Order 4.

Algorithm:

- Consider a key value K that needs to be searched.
- Searching is done from the root and then we traverse down recursively.
- If K is smaller than the root value, goto left subtree; if K is greater than the root value, search the right subtree.
- If K is found in the node, directly return the node.
- If the node does not contain K, then traverse down to the child with a greater key.
- if K is not found in the tree, return NULL.

E.g. Suppose we wish to search for index record with key value is 17.

Search will start from the root. 17 is not present here and K = 17 is less than root value = 25. We will search in left search tree. Still, K = 17 was not found. Traverse down, K = 17 is greater than value = 15. Thus, search in the right subtree of value 15. Here in the right subtree, we finally found K = 17. Thus, we will return the node.

Time complexity:
Time Complexity of the searching in B-tree = $\log_p n$ where p = order of B-tree and n = number of the search key.

Underflow and overflow in B-trees:

Underflow: If the node has too few values, i.e. has less value than what it is actually required, then this situation is known as underflow.
For roots: If number of search key values < 1., then underflow occurs.
For internal nodes: If the number of search key values < $\left\lceil \frac{P}{2} \right\rceil - 1$ then underflow occurs.

Overflow: Consider a B-tree of order p, if number of search key values in a B-tree node exceeds 'p-1', then this condition is known as overflow.
This is valid for both root nodes as well as internal nodes.

E.g.

Consider the above B-tree of order 5. If we will try to insert new node 9 then overflow occurs.

Rack Your Brain

Consider a B-tree with a search key size of 12 bytes, block size of 1 KB, data pointer is of 5 bytes and blocks pointer is of 8 bytes. Calculate the order of the B-tree?

1) 40 **2)** 41
3) 42 **4)** 43

Insertion in B-tree:
1) At level 0, a B-TREE will contain only one node known as the root node.
2) When level-0 is full, and we have to insert more entries, then the root node will split. We will get one more level as level-1.

Steps:
1) Search to determine which leaf node will hold a key.
2) If the leaf node have space, insert the key in ascending order.
3) Otherwise, split leaf node's keys into two parts and promote median key to the parent.

4) If the parent node is full, recursively split and put the median key to its parent.
5) If a promotion is made to a full root node, split and create a new root having only the promoted the median key.

SOLVED EXAMPLES

Q8 **Consider the B-tree of order 4; and insert A, B, C, D, E, F, G, H, I, J.**

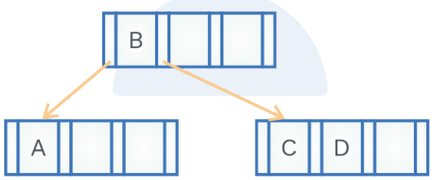
Sol: **As given, the order of B-tree =** 4.

Thus, maximum number of keys = 3.

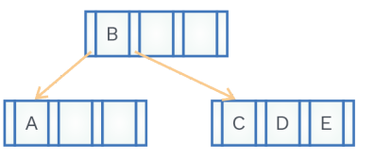
The minimum number of keys = $\left\lceil \frac{4}{2} \right\rceil - 1 = 1$

Insert A, B, C: 

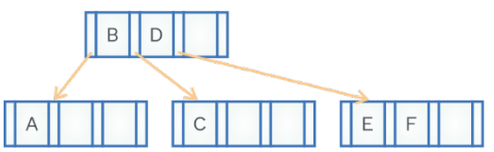
Insert D: When we try to insert D, overflow occurs. Split the leaf node's into two parts and promote the median key to the parent.



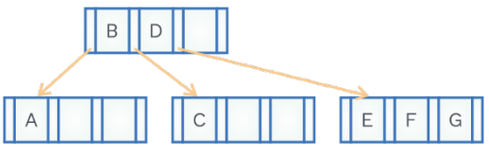
Insert E: When we try to insert E, there will be no overflow. So, just insert E.



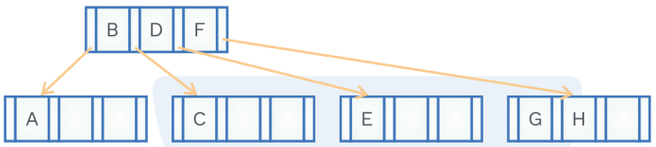
Insert F: When we will insert F into B-tree, then overflow will occur. Split the leaf node's into two parts and promote median keys to the parent.



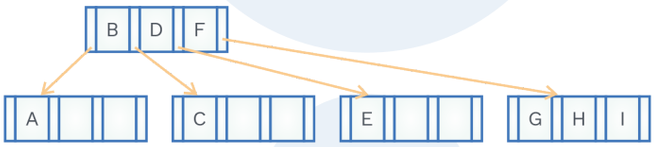
Insert G: When we will insert G, there will be no overflow as space is already available.



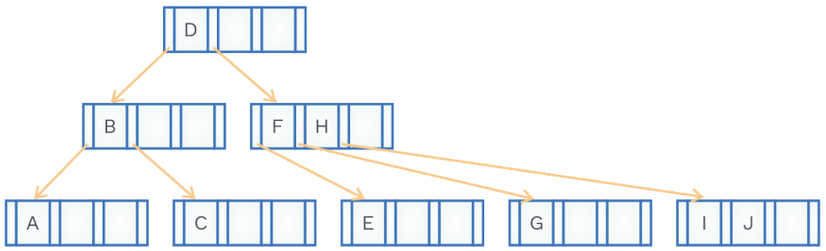
Insert H: When we will insert H into B-tree, then overflow will occur, split the leaf node into two parts and promote the median key to the parent.



Insert I: When we will insert I into B-tree, there will be no overflow.



Insert J: When we will insert J into B-tree, then overflow will occur, split the leaf node into two parts and promote the median key to the parent. Thus now, parent node is also full, so split this node also and promote the median key D to above the level.



Q9 **Consider the B-tree of order 4.**
Insert 2, 5, 10, 11, 1, 6, 9, 4, 3, 12, 18, 20, 25 into B-tree.

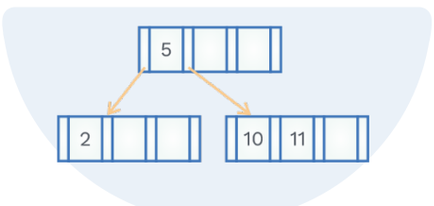
Sol: **As given, the order of B-tree = 4.**

Thus, the minimum number of keys = $\left\lceil \frac{4}{2} \right\rceil - 1 = 1$

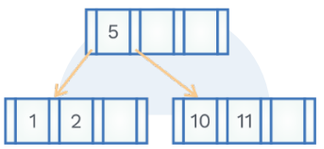
maximum number of keys = 4 – 1 = 3

Inert 2, 5, 10: 

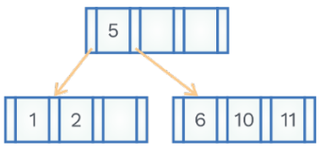
Insert 11: When we try to insert 11, there will be an overflow. Overflow splits the leaf node's into two parts and promotes the median key to the parent.



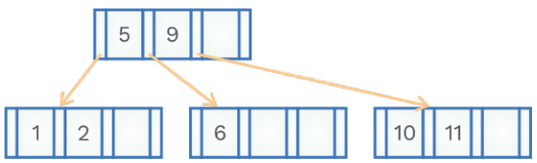
Insert 1: When we try to insert 1 into B-tree, there will be no overflow.



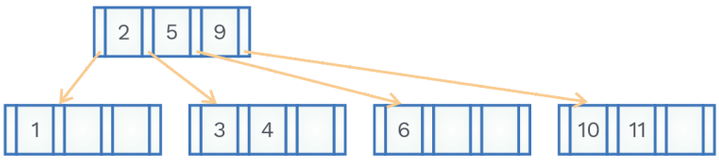
Insert 6: When we try to insert 6 into B-tree, there will be no overflow.



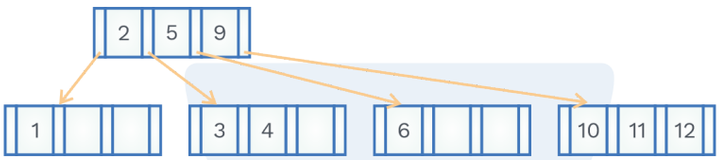
Insert 9: Leaf node will have overflow now. So split the leaf nodes into two parts and promote median key to the parent.



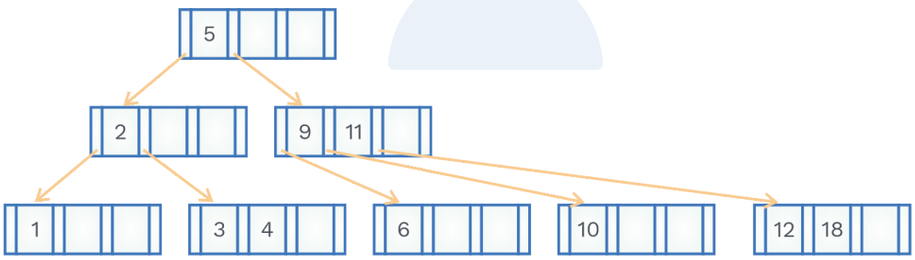
Insert 4 and 3: When we insert 4, there will be no overflow, but when we insert 3 there will be overflow and thus split the leaf node and into two parts and promote the median key, which is k = 2 to parent.



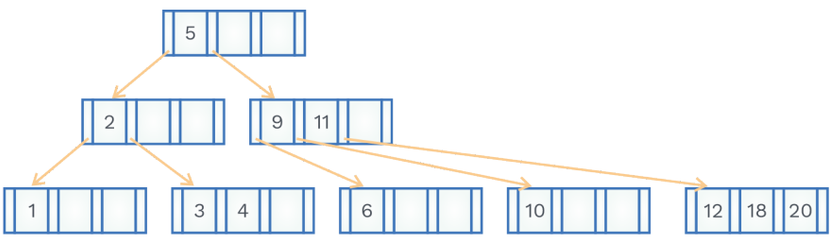
Insert 12: When we try to insert 12 then there will be no overflow.



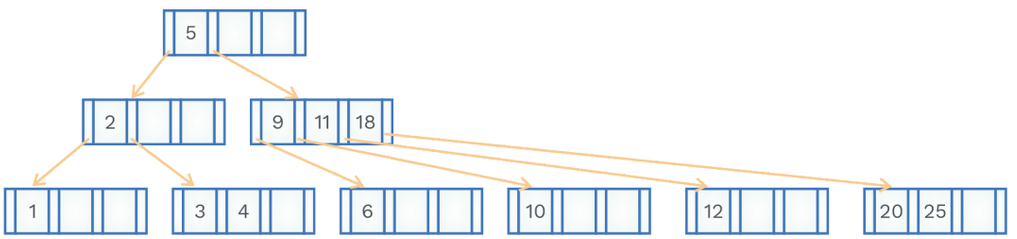
Insert 18: Now leaf node will have overflow; thus, we will split leaf node and promote 11 to the upper (parent) level.
Further, the parent level will also get overflowed. So, split nodes and promote 5 to upper level. This is shown below.



Inset 20: When we try to insert 20 into B-tree then, there will be no overflow



Insert 25: When we will insert 25, the leaf node will have an overflow, so split the leaf node and promote key value 18 to parent node.



Previous Years' Question

A B-tree used as an index for a large database table has four levels, including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are

1) 5 **2)** 4 **3)** 3 **4)** 2

Sol: Option 1) **(GATE-IT-2005)**

Deletion from B-tree:

Steps:

1) If a non-leaf key value is to be deleted, interchange it with its successor or predecessor and delete it from its new leaf node position.

2) If a leaf node block stores more than the minimum possible count of keys, simply delete the targeted key from the leaf node block. Again, if there exists a minimum count of keys in a leaf node block, the two immediate sibling leaf nodes are to be considered as described in further points.

3) If any sibling leaf node contains more than the minimum permissible count of keys, transfer the median key value to the parent node and a key from the parent to the node with deficit of keys.

4) If both of them contains exactly the minimum allowed count of keys, the deficit node is merged along with one of the siblings and a parent key.

5) If the above steps result in to less number of keys in parent node, repeat all the steps in an upward manner.

6) If this leaves the parent node with too few keys, then the process is propagated upward.

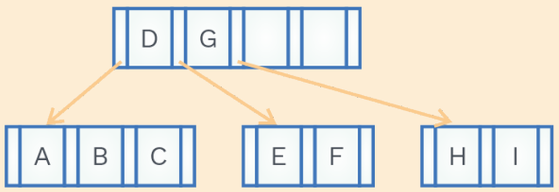
Let removing a key from a leaf node leaves l-keys in the leaf node.

i) If $l > \left\lceil p/2 \right\rceil - 1$, then we can stop, i.e. no underflow.

ii) If $l < \lceil p/2 \rceil - 1$, then this is a condition for underflow and we must rebalance the tree.

SOLVED EXAMPLES

Q10 **Consider the B-tree of order 5 given below:**

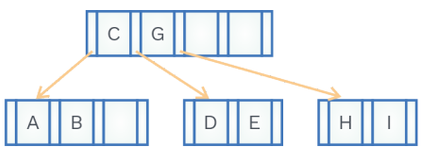


Delete F and D from the B-tree.

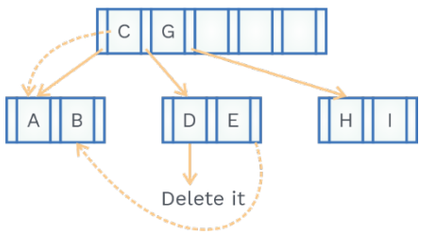
Sol: **Since the order of B-tree = 5.**

So, the maximum number of keys in leaf/internal node = 4 and the minimum number of keys in leaf/internal node = $\lceil p/2 \rceil - 1 = 2$.

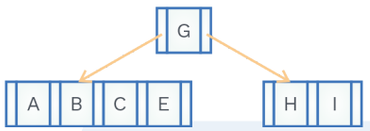
Delete F: Borrow Method; since F is present in the leaf node and the minimum number of keys which should be present in leaf node is 2. Thus, when we delete node F, we need to rebalance the B-tree. For that, we will Borrow a node from one of the siblings if the siblings have more than a minimum number of keys. Redistribute one key from this sibling to the parent node and one key from the parent to the deficient node.



ii) Delete D: Coalesce → merging method.
D is present in the leaf node, and the minimum number of keys which should present in the leaf node is 2.
Thus when we delete node D from the leaf; and both immediate siblings have exactly the minimum number of keys, we will merge deficient node with one of the sibling node, an one entry from the parent node.



⇒



5.5 B^+ TREES

A variation of B tree that is more efficient in searching for an element.

Hey learners!!

- Do you know why we use the B+ tree?
- In the B tree, the number of entries that can be present within a node is less as the structure of the B tree contains a data pointer along with the search key.
- B+ tree eliminates this drawback.
- In the B+ tree, the data pointer can be stored only at the leaf nodes.
- Thus, the structure of the leaf node and internal node both are different.
- In the B+ tree, all the key values must be present in the leaf node.
- One leaf node is linked to the other leaf node so that we can access the search key in an ordered way.
- There are some repeated search field values present in leaf node that is also present in internal nodes of the B+ tree.

Rack Your Brain

Consider a B tree having search key 12 bytes long, data pointer B bytes long, block pointer 4 bytes long, and block size of 1024 bytes. What will be the maximum number of children a node can have _____?

- **Internal node:** Consider P= order of a B+Tree.
  Structure:
  1) Internal node can be represented as :
  $<P_1, K_1, P_2, K_2, \ldots P_{m-1;}\ K_{m-1}, P_m>$ where each P_i is tree pointer.
  2) Each internal node must have, $K_1 < K_2 < K_3 \ldots < K_{m-1}$
  3) For every internal nodes, maximum P tree pointer is possible.
  4) Each internal node except the root has atleast $\lceil P/2 \rceil$ tree pointers.
  5) Internal node has $\lceil P/2 \rceil - 1$ to P – 1 search key values.
- **Root node:** The root node contains atleast 2 tree pointers.
- **Leaf node:** Structure of the leaf nodes of a B^+ tree of order P:
  $<<K_1, Pr_1>, <K_2, Pr_2>, \ldots, <K_{m-1}, Pr_{m-1}>, P_{next}>$
  where $m \leq P$ (order of tree), Pr_i = record/data pointer and P_{next} = pointer pointing to the next leaf node of B^+ - tree.

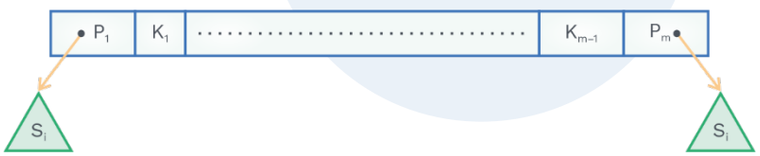


Fig. 5.11 Non-leaf Structure of a B^+ Tree

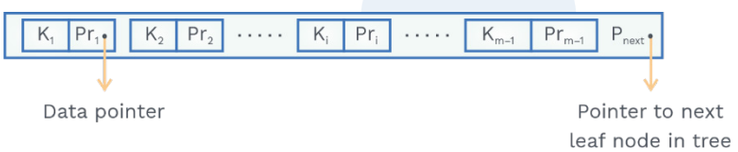


Fig. 5.12 Leaf Node Structure of a B^+ Tree

- $K_1 \leq K_2 \leq K_3 \ldots\ldots \leq K_{m-1}$ where $m \leq p$ (order of tree)
  [within each leaf node]
- Each leaf node has at least $\lceil P/2 \rceil$ values.
- All leaf nodes need to be at the same level

Note:

An internal node of a B+ tree contains more entries than a B tree.

Note:

As we got to know that the structure of leaf nodes and internal nodes are different for a B+ tree, therefore order can also be different.

- P is the order of internal nodes.
- P_{leaf} is the order for leaf nodes.
- Order of leaf represents maximum number of data pointers in a leaf node.
- Order of non-leaf (internal) node represents maximum number of children a node can have.

SOLVED EXAMPLES

Q11 **Consider the data given below for the B+ Tree:**
i) Size of the Search key field is 9 bytes long
ii) Size of block = 512 bytes
iii) Size of record pointer = 7 bytes
iv) Size of block pointer = 6 bytes
What will be the order of the leaf node and internal node?

Sol: **For internal node:**

As we know, an internal node of the B+ tree has pointers = P and search key value =(P-1)

Thus, $P * (\text{block pointer}) + (P - 1) * (\text{key field}) \leq \text{Block Size}$

$\Rightarrow P * (P_b) + (P - 1)(V) \leq \text{block size}$

$\Rightarrow P * (6) + (P - 1) * (9) \leq 512$

$\Rightarrow 15\ P \leq 521$

$\Rightarrow P = 34$

$\Rightarrow$ $O_{non-leaf}$ = Order of non-leaf (internal node)
P = 34

Structure of Leaf node

$(K_1, Pr_1)\ (K_2, Pr_2)\ \ldots\ldots\ (K_m, Pr_m)\ P_b$

$P_{leaf}\ (K + Pr) + P_b \leq \text{Block size}$

Where K = search key field
Pr = record pointer
Pb = block pointer

$\Rightarrow \quad P_{leaf}(9 + 7) + 6 \leq 512$

$\Rightarrow \quad P_{leaf} * 16 \leq 512$

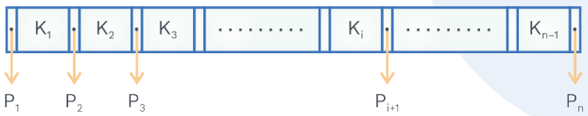
$\Rightarrow \quad P_{leaf} \leq \frac{506}{16}$

$\boxed{P_{leaf} = \text{order of leaf} = 31}$

Searching a record with key value K in B⁺ tree of order P

Algorithm:

- Start the search from the root, look for the largest key value K_i in the node, which is less than equal to key value K.
- Follow the pointer P_{i+1} to the next value until reach the leaf node.



- If K is found to be equal to K_i in the leaf, then follow Pr_i to search record.
- **Insertion in B⁺ - tree:**
- Overflow condition in B⁺ tree: When the number of search key values exceed"P-1" then this condition is overflow in B⁺ - tree.
- **Case 1:** Overflow in a leaf node, then,
- Split the leaf node into 2 nodes.
- First node will contain $\left\lceil \frac{(P-1)}{2} \right\rceil$ values where $\lceil\ \rceil$ denotes ceil function.
- Second node will contain the remaining values.
- Copy the smallest search key value of the second node to the parent node.
- **Case 2:** Overflow in non-leaf node then,
- Split the non-leaf node into two nodes.
- First node will contain $\left\lceil \frac{P}{2} \right\rceil - 1$ values/ keys.
- Move the smallest of the remaining keys to the parent.

Previous Years' Question

The order of a leaf node in a B⁺ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

1) 63 **2)** 64
3) 67 **4)** 68

Sol: Option 1) **(GATE-2007)**

Rack Your Brain

Which of the following are correct about B-tree and B⁺ tree?

S1: In a B-tree, every value of the search field appears at once at some level in the tree.

S2: An internal node in B⁺ tree with n pointers have (n + 1) search key field values.

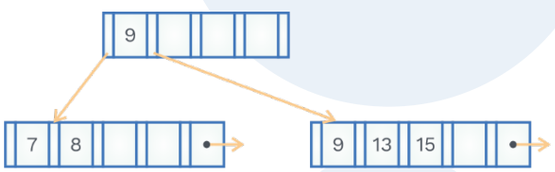
1) Only S1
2) Only S2
3) Both S1 and S2
4) Neither S1 nor S2

- Second node will contain remaining keys.

Example: Consider the B^+ tree of order 5.

Insert 8

- As the order of B^+ tree = 5. Thus, the maximum number of search keys values present inside a leaf = 4.
- When we try to insert K = 8 in the given B^+ tree, overflow occurs.
- Split the leaf nodes such that the first node contains $\left\lceil (P-1)/2 \right\rceil = 2$ key value (i.e. 7 and 8).
- Second node will contain the remaining key values (i.e. 9, 13, and 15).
- Smallest search key value of the second node will be copied to the parent node.



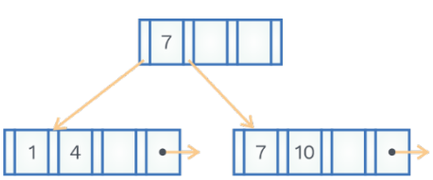
SOLVED EXAMPLES

Q12 **Construct a B^+ tree for the insertion sequence 1, 4, 7, 10, 17, 21, 31, 25 with P = 4 and P_{leaf} = 4**

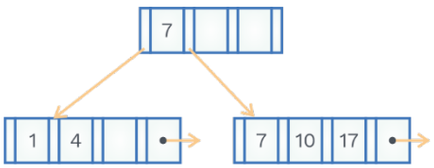
Sol: **As P = 4 and P_{leaf} = 4 is given. It means maximum number of key value present in either leaf or non-leaf node is P – 1 = 3.**

1) Insert 1, 4, 7

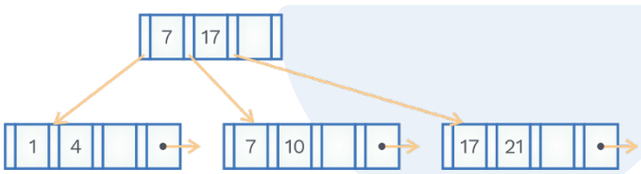
2) Insert 10 (overflow) ; $\left\lceil (P-1)/2 \right\rceil = 2$



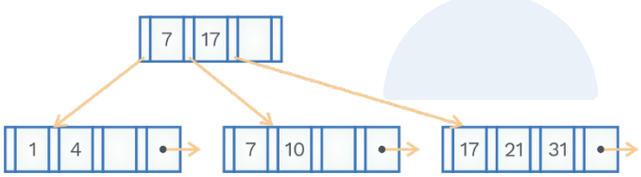
3) Insert 17 (No overflow): Insert it into the right leaf.



4) Insert 21 (overflow) calculate $\lceil (P-1)/2 \rceil$:2 split the leaf node such that the first node will contain 2 elements, and the remaining elements will be present in other node.

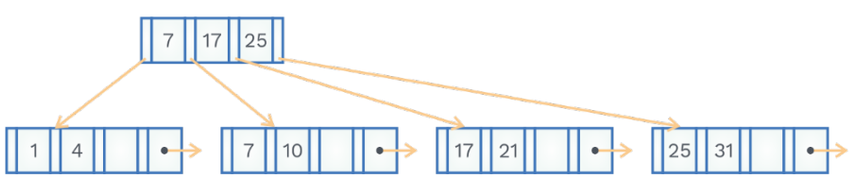


5) Insert 31 (no overflow):



6) Inert 25 (overflow): Split the leaf node such that the first node will contain

key values (i.e. 17 and 21), and the second node will contain the remaining key values (i.e. 25 and 31), and the smallest search key value i.e 25 will be copied to the parent node.



Deletion of an entry from B^+ trees:

- When we have to delete any entry from B^+ trees, first it will be deleted from the leaf node.
- If the same entry is also present in an internal node, it is required to remove that entry from here also.
- Deletion leads to the underflow situation as it will decrease the number of entries less than the required one in the leaf node.

Previous Years' Question

In a B^+ tree, if the search key value is 8 bytes long, the block size is 512 bytes and the pointer size is 2B, then the maximum order of the B^+ tree is _____

Sol: Range: 52 to 52.

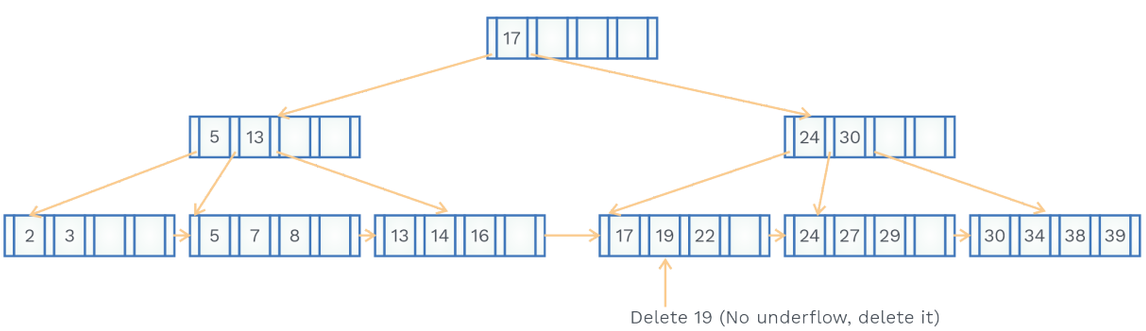
(GATE-2017 Set-2)

Steps:

- Start at root, find leaf L where entry belongs.
- Remove the entry
  - **i)** If L contains at least $\lceil P/2 \rceil - 1$ entries, done.
  - **ii)** If L has only $\lceil P/2 \rceil - 2$ entries.
    - **1)** Redistribution of entries means take borrow from the adjacent node whose parents are the same as L (also known as a sibling).
    - **2)** If redistributing fails, merge L and a sibling.
- If a merge has occurred, then corresponding entry from parent must be deleted.
- Merge could propagate to root, decreasing height.
- If the deleted entry is present in internal node, replace it with inorder successor.

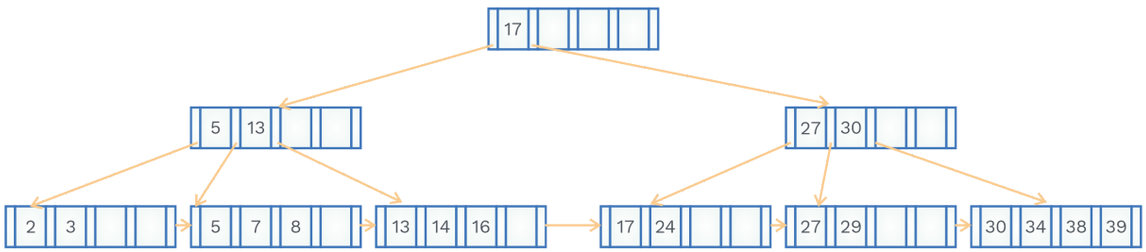
Example: Consider the B^+ tree of order P = 5. Deletion sequence: 19, 22

1) Delete 19:



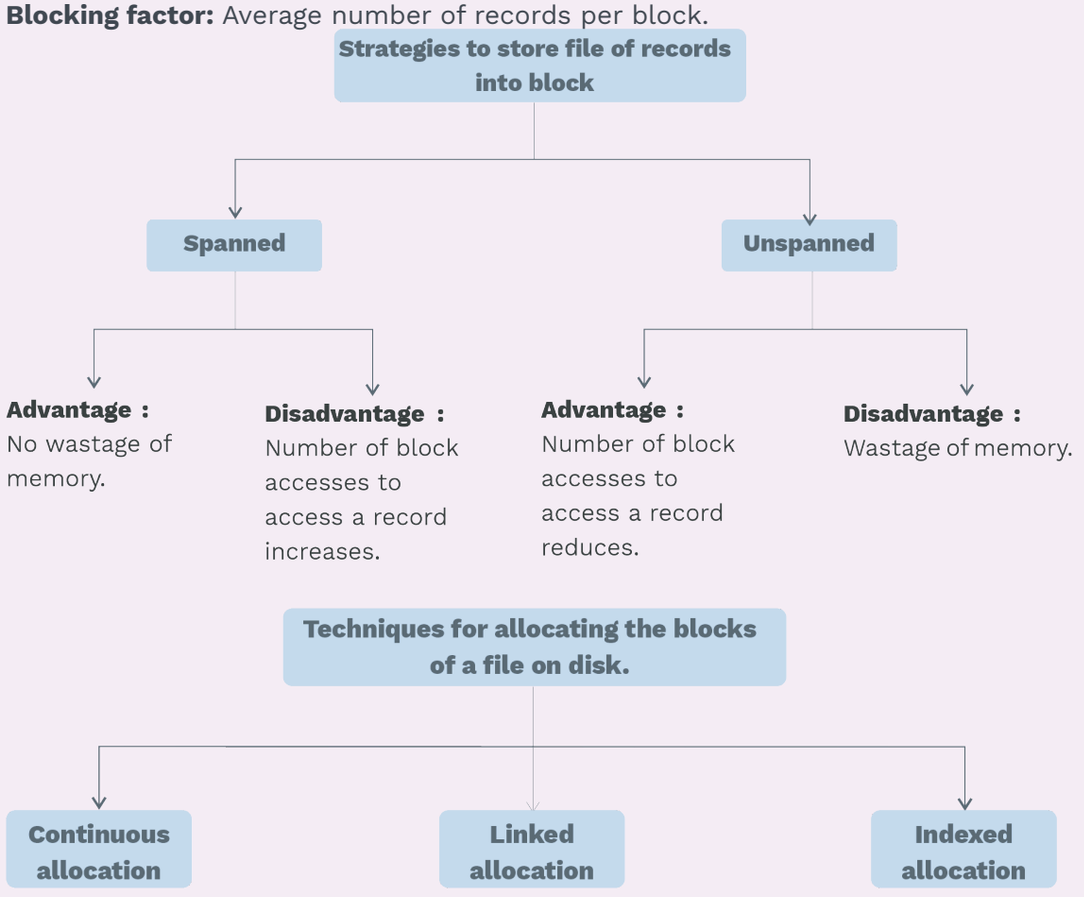
2) Delete 22:

Deleting 22 leads to underflow as $\left\lceil \frac{P}{2} \right\rceil - 1 = 2$ but leaf contains < 2 entries after deleting 22. Thus, redistribute borrowing from a sibling.



Chapter Summary

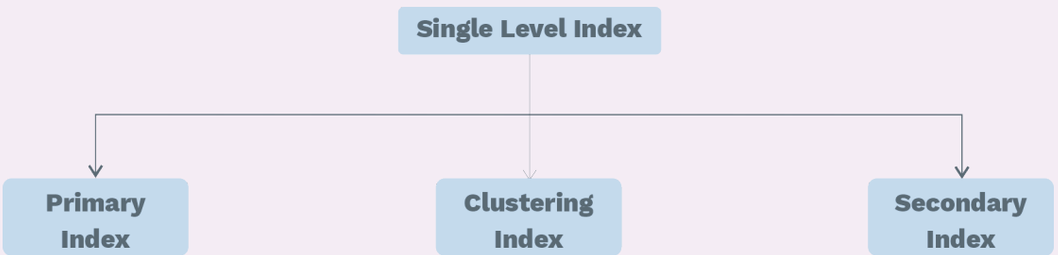
- **Files:** It is organised as a sequence of records.
- **Unordered file:** Records are placed in no particular order.
- **Blocking factor:** Average number of records per block.



- **File organization:** It refers to the organization of the data of a file into records, blocks and access structure

	Types of Organization	Access/Search Method	Average blocks to access a specific record
i)	Heap (unordered)	Linear Search (Sequential scan)	b/2
ii)	Ordered	Linear Search (Sequential scan)	b/2
iii)	Ordered	Binary Search	$\log_2 b$

- There are types of indices based on ordered files (single-level-indexes) and tree data structures (multilevel indexes, B^+ trees).



- **Anchor record:** The anchor record is the first record in each block of the data file.
- **Dense index:** If an index entry is created for every search key value, then that index is called the dense index.
- **Sparse index:** If an index is created only for some search key value, then that index is called the sparse index.
- The number of blocks accesses using clustering index $\geq \lceil \log_2 B_i \rceil + 1$ where Bi refers to the total block count.
- The secondary index may be created on an unordered field that is the non-key field or candidate key.
- Number of index entries in secondary index = number of records.
- **Multilevel index:** It is used to speed up the search operations.
- **Search tree:** It contains maximum (P-1) keys, P pointers if P is defined as order of a search tree.
- B-tree and B^+ trees are special cases of search data-structure trees.
- **B-tree:** A B-tree of order P can be defined as:

Each internal node in the B-tree is of the form

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots\ldots <K_{m-1}, Pr_{m-1}>, P_m>$$

Where m ≤ P (order of tree), Pi is the tree pointer, and Pri is a data pointer.

- All leaf nodes in B-tree are at the same level.
- **Order of tree:** the order of the tree represents the maximum number of children (Block pointer) a node can have.
- Time complexity of searching in B-tree = logpn where p = order of B-tree and n = number of the search key.