Define the following terms.

Database

- A Database is a collection of inter-related data.

DBMS (Database Management System)

- A database management system is a collection of inter-related data and set of programs to manipulate those data.
- DBMS = Database + Set of programs

Metadata

- Metadata is data about data.
- Data such as table name, column name, data type, authorized user, user access privileges for any table is called metadata for that table.

Data dictionary

- Data dictionary is an information repository which contains metadata.
- It is usually a part of the system catalog.

Data warehouse

- Data warehouse is an information repository which stored data.
- It is design to facilitate reporting and analysis.

Field

- A field is a character or group of characters that have a specific meaning.
- It is also called a data item. It is represented in the database by a value.
- For Example customer id, name, society and city are all fields for customer Data.

Record

- A record is a collection of logically related fields.
- For examples, collection of fields (id, name, society & city) forms a record for customer.

Explain disadvantages of file system (file processing systems) compare to Database management system. OR Explain disadvantages of conventional file-based system compared to Database management system.

Data Redundancy

- It is possible that the same information may be duplicated in different files. This leads to data redundancy.
- Data redundancy results in memory wastage.
- For example, consider that some customers have both kinds of accounts - saving and current. In this case, data about customers - name, address, e-mail, contact number - will be duplicated in both files, file for saving accounts and file for current accounts. This leads to requirement of higher storage space. In other words, same information will be stored in two different locations (files). And, it wastes memory.

Data Inconsistency

- Due to data redundancy, it is possible that data may not be in consistent state.
- For example, consider that an address of some customer changes. And, that customer has both kinds of accounts. Now, it is possible that this changed address is updated in only one file, leaving address in other file as it is. As a result of this, same customer will have two different addresses in two different files, making data inconsistent.

Difficulty in Accessing Data

- Accessing data is not convenient and efficient in file processing system.

- For example, suppose, there is a program to find information about all customers. But, what if there is a need to find out all customers from some particular city. In this case, there are two choices here: One, find out all customers using available program, and then extract the needed customers manually. Second, develop new program to get required information. Both options are not satisfactory.
- For each and every different kind of data access, separate programs are required. This is neither convenient nor efficient.

Limited Data Sharing

- Data are scattered in various files.
- Different files may have different formats. And these files may be stored in different folders (directories) may be of different computers of different departments.
- So, due to this data isolation, it is difficult to share data among different applications.

Integrity Problems

- Data integrity means that the data contained in the database is both correct and consistent. For this purpose, the data stored in database must satisfy certain types of constraints (rules).
- For example, a balance for any account must not be less than zero. Such constraints are enforced in the system by adding appropriate code in application programs. But, when new constraints are added, such as balance should not be less than Rs. 5000, application programs need to be changed. But, it is not an easy task to change programs whenever required.

Atomicity Problems

- Any operation on database must be atomic. This means, operation completes either 100% or 0%.
- For example, a fund transfer from one account to another must happen in its entirely. But, computer systems are vulnerable to failure, such as system crash, virus attack. If a system failure occurs during the execution of fund transfer operation, it may possible that amount to be transferred, say, Rs. 500, is debited from one account, but is not credited to another account.
- This leaves database in inconsistent state. But, it is difficult to ensure atomicity in a file processing system.

Concurrent Access Anomalies

- Multiple users are allowed to access data simultaneously (concurrently). This is for the sake of better performance and faster response.
- Consider an operation to debit (withdrawal) an account. The program reads the old balance, calculates the new balance, and writes new balance back to database. Suppose an account has a balance of Rs. 5000. Now, a concurrent withdrawal of Rs. 1000 and Rs. 2000 may leave the balance Rs. 4000 or Rs. 3000 depending upon their completion time rather than the correct value of Rs. 2000.
- Here, concurrent data access should be allowed under some supervision.
- But, due to lack of co-ordination among different application programs, this is not possible in file processing systems.

Security Problems

- Database should be accessible to users in a limited way.
- Each user should be allowed to access data concerning his application only.
- For example, a customer can check balance only for his/her own account. He/She should not have access to information about other accounts.
- But, in file processing system, application programs are added in an ad hoc manner by different programmers. So, it is difficult to enforce such kind of security constraints.

Explain advantages (benefits) of DBMS over file management system. OR Explain purpose of database system.

Minimal Data Redundancy (Duplication)

- Due to centralized database, it is possible to avoid unnecessary duplication of information.
- This leads to reduced data redundancy.
- It prevents memory wastage.
- It also reduced extra processing time to get required data.

Shared Data

- All authorized user and application program can share database easily.

Data Consistency

- Data inconsistency occurs due to data redundancy.
- With reduced data redundancy such type of data inconsistency can be eliminated.
- This results in improved data consistency.

Data Access

- DBMS utilizes a variety of techniques to retrieve data.
- Required data can be retrieved by providing appropriate query to the DBMS.
- Thus, data can be accessed in convenient and efficient manner.

Data Integrity

- Data in database must be correct and consistent.
- So, data stored in database must satisfy certain types of constraints (rules).
- DBMS provides different ways to implement such type of constraints (rules).
- This improves data integrity in a database.

Data Security

- Database should be accessible to user in a limited way.
- DBMS provides way to control the access to data for different user according to their requirement.
- It prevents unauthorized access to data.
- Thus, security can be improved.

Concurrent Access

- Multiple users are allowed to access data simultaneously.
- Concurrent access to centralized data can be allowed under some supervision.
- This result in better performance of system and faster response.

Guaranteed Atomicity

- Any operation on database must be atomic. This means, operation must be executed either 100% or 0%.
- This type of atomicity is guaranteed in DBMS.

List and explain the applications of DBMS.

Airlines and railways

- Airlines and railways use online databases for reservation, and for displaying the schedule information.

Banking

- Banks use databases for customer inquiry, accounts, loans, and other transactions.

Education

- Schools and colleges use databases for course registration, result, and other information.

Telecommunications

- Telecommunication departments use databases to store information about the communication network, telephone numbers, record of calls, for generating monthly bills, etc.

Credit card transactions

- Databases are used for keeping track of purchases on credit cards in order to generate monthly statements.

E-commerce

- Integration of heterogeneous information sources (for example, catalogs) for business activity such as online shopping, booking of holiday package, consulting a doctor, etc.

Health care information systems and electronic patient record

- Databases are used for maintaining the patient health care details in hospitals.

Digital libraries and digital publishing

- Databases are used for management and delivery of large bodies of textual and multimedia data.

Finance

- Databases are used for storing information such as sales, purchases of stocks and bonds or data useful for online trading.

Sales

- Databases are used to store product, customer and transaction details.

Human resources

- Organizations use databases for storing information about their employees, salaries, benefits, taxes, and for generating salary checks.

Describe functions (responsibility, roles, and duties) of DBA to handle DBMS.

DBA

- The full name of DBA is Database Administrator.
- Database Administrator is a person in the organization who controls the design and the use of database.

Functions or Responsibilities of DBA are as under:

Schema Definition

- DBA defines the logical schema of the database.
- A schema refers to the overall logical structure of the database.
- According to this schema, database will be designed to store required data for an organization.

Storage Structure and Access Method Definition

- DBA decides how the data is to be represented in the database.
- Based on this, storage structure of the database and access methods of data is defined.

Defining Security and Integrity Constraints

- DBA decides various security and integrity constraints.
- DDL provides facilities to specifying such constraints.

Granting of Authorization for Data Access

- The DBA determines which user needs access to which part of the database.
- According to this, various types of authorizations (permissions) are granted to different users.
- This is required to prevent unauthorized access of a database.

Liaison with Users

- DBA is responsible to provide necessary data to user.
- User should be able to write the external schema, using DDL.

Assisting Application Programmers

- DBA provide assistance to application programmers to develop application programs.

Monitoring Performance

- The DBA monitors performance of the system.
- The DBA ensure that better performance is maintained by making change in physical or logical schema if required.

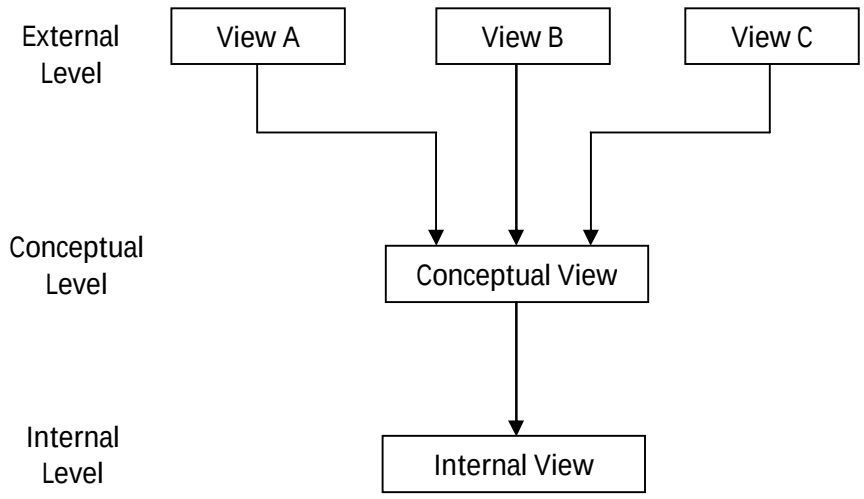
Backup and Recovery

- Database should not be lost or damaged.
- The task of DBA is to backing up the database on some storage devices such as DVD, CD or Magnetic Tape or remote servers.
- In case of failures, such as flood or virus attack, Database is recovered from this backup.

Explain three levels ANSI SPARC Database System. OR Explain three level Data abstraction.

The ANSI SPARC architecture divided into three levels:

1) External level
2) Conceptual level
3) Internal level



Internal Level

- This is the lowest level of the data abstraction.
- It describes **how** the data are actually stored on storage devices.
- It is also known as a physical level.
- The internal view is described by internal schema.
- Internal schema consists of definition of stored record, method of representing the data field and access method used.

Conceptual Level

- This is the next higher level of the data abstraction.
- It describes **what** data are stored in the database and what relationships exist among those data.
- It is also known as a logical level.
- Conceptual view is defined by conceptual schema. It describes all records and relationship.

External Level

- This is the highest level of data abstraction.
- It is also known as view level.
- It describes only part of the entire database that a particular end user requires.
- External view is describes by external schema.
- External schema consists of definition of logical records, relationship in the external view and method of deriving the objects from the conceptual view.
- This object includes entities, attributes and relationship.

Explain Mapping. OR
Explain external and internal mapping. OR
What is mapping? Describe type of mapping.

Mapping

- The process of transforming requests and results between the three levels is called mapping.

Types of Mapping

- Conceptual/Internal Mapping
- External/Conceptual Mapping

Conceptual / Internal Mapping

- It relates conceptual schema with internal schema.
- It defines correspondence between the conceptual schema and the database stored in physical devices.
- It specifies how conceptual records and fields are presented at the internal level.
- If the structure of stored database is changed, then conceptual/internal mapping must be changed accordingly and conceptual schema can remain invariant.
- There could be one mapping between conceptual and internal levels.

External / Conceptual Mapping

- It relates each external schema with conceptual schema.
- It defines correspondence between a particular external view and conceptual schema.
- If the structure of conceptual schema is changed, then external/conceptual mapping must be changed accordingly and external schema can remain invariant.
- There could be several mappings between external and conceptual levels.

Explain Data Independence.

Data Independence

- Data independency is the ability to modify a schema definition in one level without affecting a schema definition in the next higher level.

Types of data independence

- Physical data independence
- Logical data independence

Physical data independence

- Physical data independence allows changing in physical storage devices or organization of file without change in the conceptual view or external view.
- Modifications at the internal level are occasionally necessary to improve performance.
- Physical data independence separates conceptual level from the internal level.
- It is easy to achieve physical data independence.

Logical data independence

- Logical data independence is the ability to modify the conceptual schema without requiring any change in application programs.
- Conceptual schema can be changed without affecting the existing external schema.
- Modifications at the logical level are necessary whenever the logical structure of the database is altered.
- Logical data independence separates external level from the conceptual view.
- It is difficult to achieve logical data independence.

Explain different database users.

There are four different database users.

Application programmers

- These users are computer professionals who write application programs using some tools.

Sophisticated users

- These users interact with system without writing program. They form their request in a database query language.

Specialized users

- These users write specialized database applications that do not fit into the traditional data processing framework.

Naive users

- These users are unsophisticated users who have very less knowledge of database system.
- These users interact with the system by using one of the application programs that have been written previously.
- Examples, people accessing database over the web, bank tellers, clerical staff etc.

Differentiate the DA and DBA.

DA (Data Administrator)	DBA (Database Administrator)
The data administrator is a person in the organization who controls the data of the database.	The database administrator is a person in the organization who controls the design and the use of the database.
DA determines what data to be stored in database based on requirements of the organization.	DBA provides necessary technical support for implementing a database.
DA is involved more in the requirements gathering, analysis, and design phases.	DBA is involved more in the design, development, testing and operational phases.
DA is a manager or some senior level person in an organization who understands organizational	DBA is a technical person having knowledge of database technology.

requirements with respect to data.	
DA does not need to be a technical person, but any kind of knowledge about database technology can be more beneficiary.	DBA does not need to be a business person, but any kind of knowledge about a functionality of an organization can be more beneficiary.
DA is a business focused person, but, he/she should understand more about the database technology.	DBA is a technically focused person, but, he/she should understand more about the business to administer the databases effectively.

Explain Database System Architecture.

Components of a DBMS

These functional units of a database system can be divided into two parts:

1. Query Processor Units (Components)
2. Storage Manager Units

Query Processor Units:

Query processor units deal with execution of DDL and DML statements.

- **DDL Interpreter —** Interprets DDL statements into a set of tables containing metadata.
- **DML Compiler —** Translates DML statements into low level instructions that the query evaluation engine understands.
- **Embedded DML Pre-compiler —** Converts DML statements embedded in an application program into normal procedure calls in the host language.
- **Query Evaluation Engine —** Executes low level instructions generated by DML compiler.

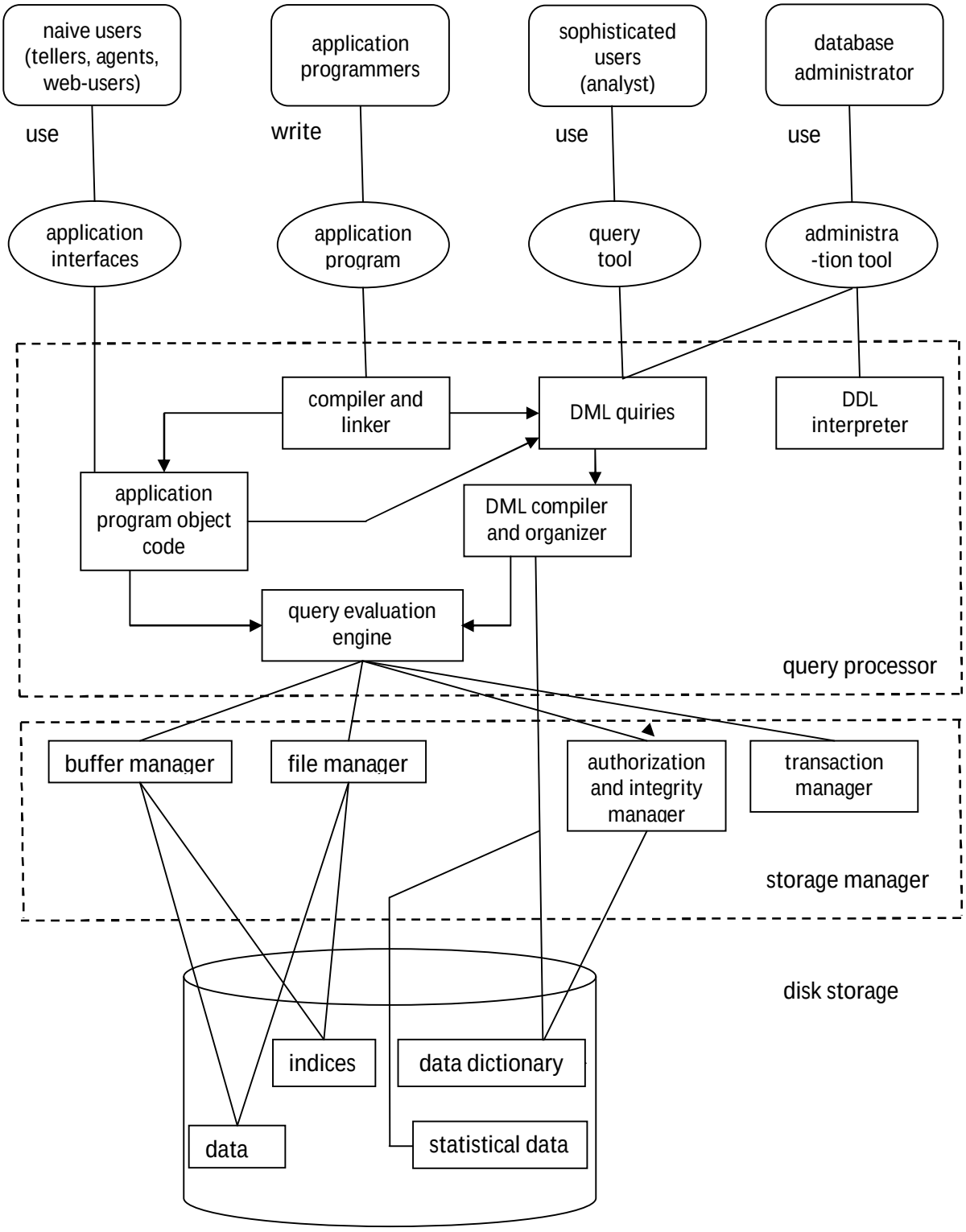
Storage Manager Units:

Storage manager units provide interface between the low level data stored in database and the application programs & queries submitted to the system.

- **Authorization Manager** — Checks the authority of users to access data.
- **Integrity Manager** — Checks for the satisfaction of the integrity constraints.
- **Transaction Manager** — Preserves atomicity and controls concurrency.
- **File Manager** — Manages allocation of space on disk storage.
- **Buffer Manager** — Fetches data from disk storage to memory for being used.

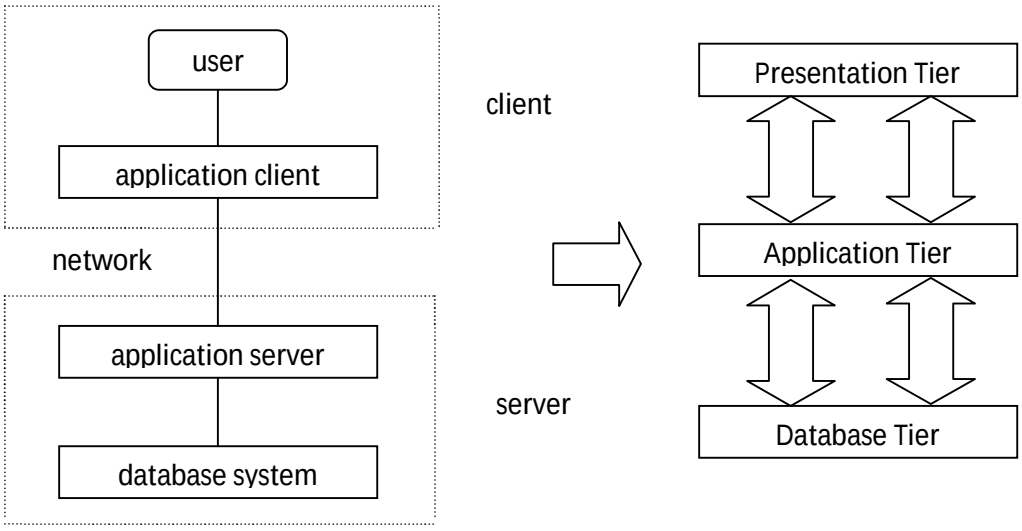
In addition to these functional units, several data structures are required to implement physical storage system. These are described below:

- **Data Files** — To store user data.
- **Data Dictionary and System Catalog** — To store metadata. It is used heavily, almost for each and every data manipulation operation. So, it should be accessed efficiently.
- **Indices** — To provide faster access to data items.
- **Statistical Data** — To store statistical information about the data in the database. This information is used by the query processor to select efficient ways to execute a query.


naive users (tellers, agents, web-users)
application programmers
sophisticated users (analyst)
database administrator
use
write
use
use
application interfaces
application program
query tool
administra -tion tool
compiler and linker
DML quiries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

Explain database system 3 tier architecture with clear diagram in detail.

- Most widely used architecture is 3-tier architecture.
- 3-tier architecture separates it tier from each other on basis of users.



Database (Data) Tier

- At this tier, only database resides.
- Database along with its query processing languages sits in layer-3 of 3-tier architecture.
- It also contains all relations and their constraints.

Application (Middle) Tier

- At this tier the application server and program, which access database, resides.
- For a user this application tier works as abstracted view of database.
- Users are unaware of any existence of database beyond application.
- For database-tier, application tier is the user of it.
- Database tier is not aware of any other user beyond application tier.
- This tier works as mediator between the two.

User (Presentation) Tier

- An end user sits on this tier.
- From a users aspect this tier is everything.
- He/she doesn't know about any existence or form of database beyond this layer.
- At this layer multiple views of database can be provided by the application.
- All views are generated by an application, which resides in application tier.

Explain keys.

Super key

- A super key is a set of one or more attributes that allow us to identify each tuple uniquely in a relation.
- For example, the enrollment_no of a student is sufficient to distinguish one student tuple from another.

Candidate key

- Candidate key is a super key for which no proper subset is a super key.
- For example, combination of roll_no and department_name is sufficient to distinguish one student tuple from another. But either roll_no or department_name alone is not sufficient to distinguish one student tuple from another. So {roll_no, department_name} is candidate key.
- Combination of enrollment_no and department_name is sufficient to distinguish one student tuple from another. But enrollment_no alone is also sufficient to distinguish one student tuple from another. So {enrollment_no, department_name} is not candidate key.

Primary key

- A Primary key is a candidate key that is chosen by database designer to identify tuples uniquely in a relation.

Alternate key

- An Alternate key is a candidate key that is not chosen by database designer to identify tuples uniquely in a relation.

Foreign key

- A foreign key is a set of one or more attributes whose values are derived from the primary key attribute of another relation.

What is relational algebra? Explain relational algebraic operation.

- Relational algebra is a language for expressing relational database queries.
- Relation algebra is a procedural query language.
- Relational algebraic operations are as follows:

Selection:-

- **Operation:** Selects tuples from a relation that satisfy a given condition.
  It is used to select particular tuples from a relation.
  It selects particular tuples but all attribute from a relation.
- **Symbol:** σ (Sigma)
- **Notation:** $\sigma_{(condition)}$ <Relation>
- **Operators:** The following operators can be used in a condition.
  =, ?, <, >, <=,>=, Λ(AND), V(OR)
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out all the students of CE department.

  $\sigma_{Dept="CE"}$ (Student)
- **Output**: The above query returns all tuples which contain CE as department name.
  Output of above query is as follows

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
109	Amit	CE	7
133	Anita	CE	9

Projection:-

- **Operation:** Selects specified attributes of a relation.
  It selects particular attributes but all tuples from a relation.
- **Symbol:** $\prod$ (Pi)
- **Notation:** $\prod_{(attribute\ set)}$ <Relation>
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: List out all students with their roll no, name and department name.

  $\prod_{Rno,\ Name,\ Dept}$ (Student)
- **Output**: The above query returns all tuples with three attributes roll no, name and department name.
  Output of above query is as follows

Student		
Rno	**Name**	**Dept**
101	Ramesh	CE
108	Mahesh	EC
109	Amit	CE
125	Chetan	CI
138	Mukesh	ME
128	Reeta	EC
133	Anita	CE

- **Example**: List out all students of CE department with their roll no, name and department name.

  $\prod_{Rno,\ Name,\ Dept}$ ($\sigma_{Dept="CE"}$ (Student))

- **Output**: The above query returns tuples which contain CE as department with three attributes roll no, name and department name.

  Output of above query is as follows

Student		
Rno	**Name**	**Dept**
101	Ramesh	CE
109	Amit	CE
133	Anita	CE

Division:-

- **Operation:** The division is a binary relation that is written as R÷S.

  The result consists of the header of R but not in the header of S, for which it holds that all the tuples in S are presented in R.
- **Symbol:** ÷
- **Notation:** R ÷ S
- Consider following table

Project
Task
Database1
Database2

Work	
Student	**Task**
Shah	Database1
Shah	Database2
Shah	Compiler1
Vyas	Database1
Vyas	Compiler1
Patel	Database1
Patel	Database2

- **Example**: Find out all students having both tasks Database1 as well as Database 2.

  $\prod_{(student)}(Work) \div \prod_{(Task)}(Project)$
- **Output**: It gives name of all students whose task is both Database1 as well as Database 2.

  Output of above query is as follows

Student
Shah
Patel

Cartesian product:-

- **Operation:** Combines information of two relations.

  It will multiply each tuples of first relation to each tuples of second relation.

  It is also known as Cross product operation and similar to mathematical Cartesian product operation.
- **Symbol:** X (cross)
- **Notation:** Relation1 X Relation2

- **Resultant Relation :**
  - ✓ If relation1 and relation2 have n1 and n2 attributes respectively, then resultant relation will have n1 + n2 attributes from both the input relations.
  - ✓ If both relations have some attribute having same name, it can be distinguished by combing relation-name.attribute-name.
  - ✓ If relation1 and relation2 have n1 and n2 tuples respectively, then resultant relation will have n1*n2 attributes, combining each possible pair of tuples from both the input relations.

R

A	1
B	2
D	3

S

A	1
D	2
E	3

R × S

A	1	A	1
A	1	D	2
A	1	E	3
B	2	A	1
B	2	D	2
B	2	E	3
D	3	A	1
D	3	D	2
D	3	E	3

- Consider following table

Emp		
Empid	**Empname**	**Deptname**
S01	Manisha	Finance
S02	Anisha	Sales
S03	Nisha	Finance

Dept	
Deptame	**Manager**
Finance	Arun
Sales	Rohit
Production	Kishan

- **Example**:

Emp × Dept				
Empid	**Empname**	**Emp.Deptname**	**Dept.Deptname**	**Manager**
S01	Manisha	Finance	Finance	Arun
S01	Manisha	Finance	Sales	Rohit
S01	Manisha	Finance	Production	Kishan
S02	Anisha	Sales	Finance	Arun
S02	Anisha	Sales	Sales	Rohit
S02	Anisha	Sales	Production	Kishan
S03	Nisha	Finance	Finance	Arun
S03	Nisha	Finance	Sales	Rohit
S03	Nisha	Finance	Production	Kishan

Join:-

Natural Join Operation (⋈)

- **Operation:** Natural join will retrieve information from multiple relations. It works in three steps.
  1. It performs Cartesian product
  2. Then it find consistent tuples and inconsistent tuples are deleted
  3. Then it delete duplicate attributes

- **Symbol:** ⋈
- **Notation:** Relation1 ⋈ Relation2
- Consider following table

Emp		
Empid	**Empname**	**Deptname**
S01	Manisha	Finance
S02	Anisha	Sales
S03	Nisha	Finance

Dept	
Deptame	**Manager**
Finance	Arun
Sales	Rohit
Production	Kishan

- **Example**:

Emp ⋈ Dept			
Empid	**Empname**	**Deptname**	**Manager**
S01	Manisha	Finance	Arun
S02	Anisha	Sales	Rohit
S03	Nisha	Finance	Arun

The Outer Join Operation

- An outer join does not require each record in the two joined tables to have a matching record.
- In natural join some records are missing if we want that missing records than we have to use outer join.
- The outer join operation can be divided into three different forms:
  1. Left outer join (⟕)
  2. Right outer join (⟖)
  3. Full outer join (⟗)
- Consider following tables

College		
Name	**Id**	**Department**
Manisha	S01	Computer
Anisha	S02	Computer
Nisha	S03	I.T.

Hostel		
Name	**Hostel_name**	**Room_no**
Anisha	Kaveri hostel	K01
Nisha	Godavari hostel	G07
Isha	Kaveri hostel	K02

Left outer join (⟕)

- The left outer join retains all the tuples of the left relation even through there is no matching tuple in the right relation.
- For such kind of tuples having no matching, the attributes of right relation will be padded with null in resultant relation.
- Example : College ⟕ Hostel

College ⟕ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Manisha	S01	Computer	Null	Null
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07

Right outer join (⟖)

- The right outer join returns all the tuples of the right relation even though there is no matching tuple in the left relation.
- For such kind of tuple tuples having no matching, the attributes of left relation will be padded with null in resultant relation.
- Example : College ⟖ Hostel

College ⟖ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07
Isha	Null	Null	Kaveri Hostel	K02

Full outer join (⟗)

- The full outer join returns all the tuples of both of the relations. It also pads null values whenever required.
- Example : College ⟗ Hostel

College ⟗ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Manisha	S01	Computer	Null	Null
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07
Isha	Null	Null	Kaveri Hostel	K02

Union

- **Operation:** Selects tuples those are in either or both of the relations.
- **Symbol :** U(union)
- **Notation :** Relation1 U Relation2
- **Requirement:** Union must be taken between compatible relations.
  Relations R and S are compatible, if
  - ✓ Both have same arity, i.e. total numbers of attributes, and
  - ✓ Domains of i^{th} attribute of R and S are same type.
- **Example :**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R U S

A	1
B	2
C	2
D	3
F	4
E	5
E	4

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

Emp U Cst	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha
4	Isha

Intersection

- **Operation:** Selects tuples those are in both relations.
- **Symbol :** ∩ (intersection)
- **Notation :** Relation1 ∩ Relation2
- **Requirement:** Set-intersection must be taken between compatible relations.
  Relations R and S are compatible, if
  - ✓ Both have same arity, i.e. total numbers of attributes, and
  - ✓ Domains of i^{th} attributes of R and S are same type.
- **Example**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R ∩ S

A	1
D	3

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

Emp ∩ Cst	
Id	**Name**
1	Manisha
2	Anisha

Difference

- **Operation:** Selects tuples those are in first (left) relation but not in second (right) relation.
- **Symbol : —** (minus)

- **Notation :** Relation1 — Relation2
- **Requirement:** Set-difference must be taken between compatible relations.
  Relations R and S are compatible, if
  - ✓ Both have same arity, i.e. total numbers of attributes, and
  - ✓ Domains of i^{th} attribute of R and S are same type.
- **Example :**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R — S

B	2
F	4
E	5

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

Emp — Cst	
Id	**Name**
3	Nisha

Cst — Emp	
Id	**Name**
4	Isha

Rename:-

- **Operation:** It is used to rename a relation or attributes.
- **Symbol:** ρ (rho)
- **Notation:** $\rho_A(B)$ Rename relation B to A.

  $\rho_{A(X1,X2....Xn)}(B)$ Rename relation B to A and its attributes to X1, X2, ..., Xn.
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out highest CPI from student table.

$$\Pi_{CPI}(Student) - \Pi_{A.CPI}(\sigma_{A.CPI<B.CPI}(\rho_A(Student) \times \rho_B(Student)))$$

- **Output**: The above query returns highest CPI.

Output of above query is as follows

CPI
9

Aggregate Function:-

- **Operation:** It takes a more than one value as input and returns a single value as output (result).
- **Symbol:** G
- **Notation:** G function (attribute) (relation)
- **Aggregate functions:** Sum, Count, Max, Min, Avg.
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out sum of all students CPI.

G sum (CPI) (Student)

Output: The above query returns sum of CPI.

Output of above query is as follows

sum
51

- **Example**: Find out max and min CPI.

G max (CPI), min (CPI) (Student)

Output: The above query returns sum of CPI.

Output of above query is as follows

max	min
9	6

Consider following schema and represent given statements in relation algebra form.

*** Branch(branch_name,branch_city)**

*** Account(branch_name, acc_no, balance)**

***Depositor(customer_name, acc_no)**

(i) Find out list of customer who have account at 'abc' branch.

$\prod_{customer\_name} (\sigma_{branch\_name="abc"} (Depositor \bowtie Account))$

(ii) Find out all customer who have account in 'Ahmedabad' city and balance is greater than 10,000.

$\prod_{customer\_name} (\sigma\ Branch._{branch\_city="Ahmedabad"} \wedge \sigma\ Account._{balance\ >10000} (Branch \bowtie Account \bowtie Depositor))$

(iii) find out list of all branch name with their maximum balance.

$\prod_{branch\_name\ ,\ G\ max\ (balance)} (Account)$

Define the following terms.

Entity

- An entity is a thing or object or person in the real world that is distinguishable from all other object.
- E.g. book, student, employee, college etc…

Entity sets

- An entity set is a set of entities of same type that share the same properties or attributes.
- E.g. the set of all students in a college can be defined as entity set student.

Relationship

- Relationship is an association (connection) between several entities.
- Relationship between 2 entities is called binary relationship.
- E.g. book is issued by student where book and student are entities and issue is relation.

Relationship set

- Relationship set is a set of relationships of the same type.

Attributes

- Attributes are properties hold by each member of an entity set.
- E.g. entity is student and attributes of students are enrollmentno, name, address, cpi etc …

Types of attributes

- **Simple attribute:** It cannot be divided into subparts. E.g. cpi, rollno
  Composite attribute: It can be divided into subparts. E.g. address, name
- **Single valued attribute:** It has single data value. E.g. enrollmentno, birthdate
  Multi valued attribute: It has multiple data value. E.g. phoneno (may have multiple phones)
- **Stored attribute:** It's value is stored manually in database. E.g. birthdate
- **Derived attribute**: It's value is derived or calculated from other attributes. E.g. age (can be calculated using current date and birthdate).

Descriptive attributes

- A relationship may also have attributes like an entity. These attributes are called descriptive attributes.
- E.g. Student gets degree certificate on 14th March 2011. Student and Degree are entities, gets certificate is relation and certificate date is an attribute of relationship.

Recursive relationship set

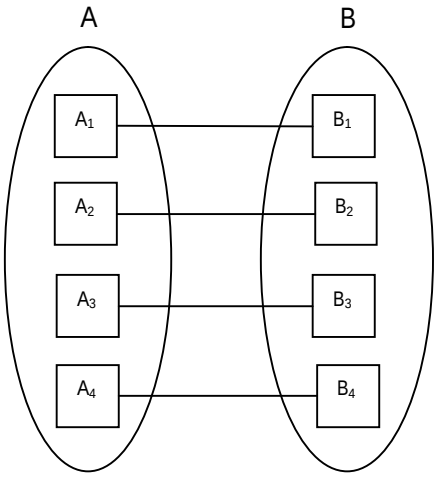
- The same entity set participates in a relationship set more than once then it is called recursive relationship set.
- E.g. an employee entity participated in relationship under with department entity as an employee as well manager also.

Explain various mapping cardinality (cardinality constraint).

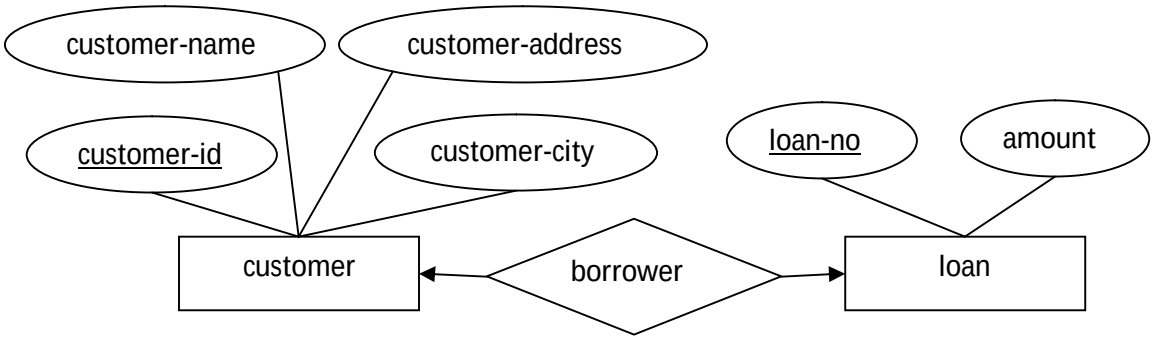
Mapping cardinality (cardinality constraints)

- It represents the number of entities of another entity set which are connected to an entity using a relationship set.
- It is most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - ✓ One to one
  - ✓ One to many
  - ✓ Many to one
  - ✓ Many to many

One-to-one relationship

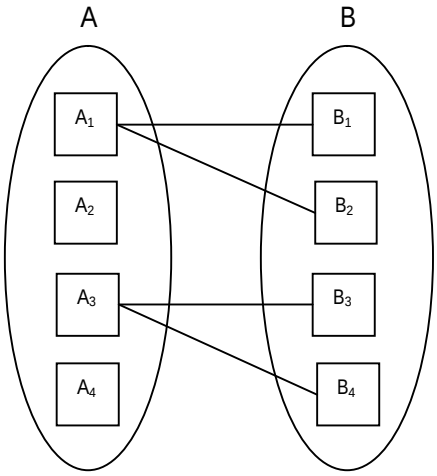


- An entity in A is associated with at most (only) one entity in B and an entity in B is associated with at most (only) one entity in A.

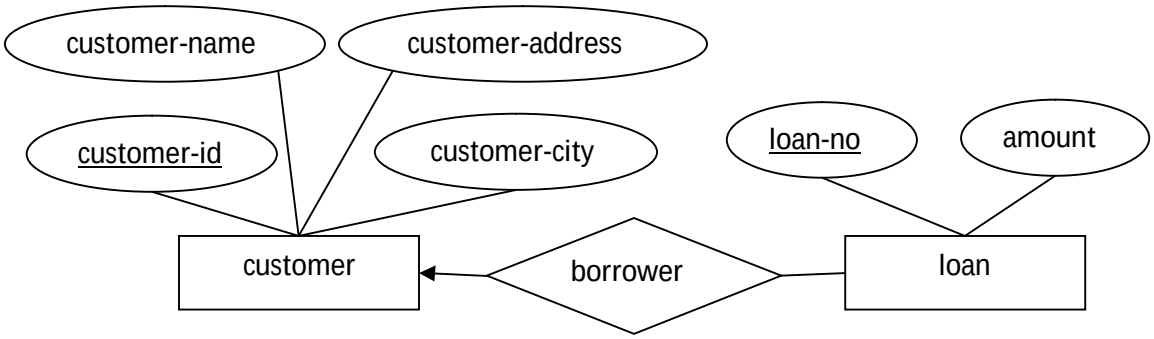


- A customer is connected with only one loan using the relationship borrower and a loan is connected with only one customer using borrower.

One-to-many relationship

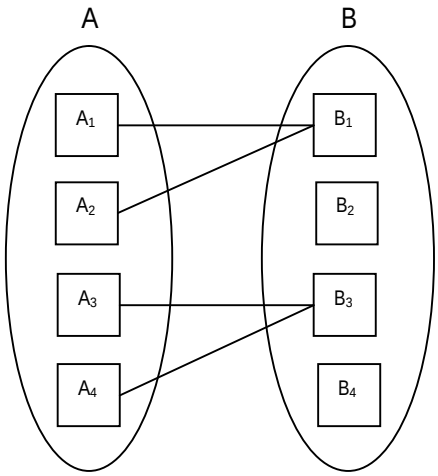


- An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with at most one (only) entity in A.

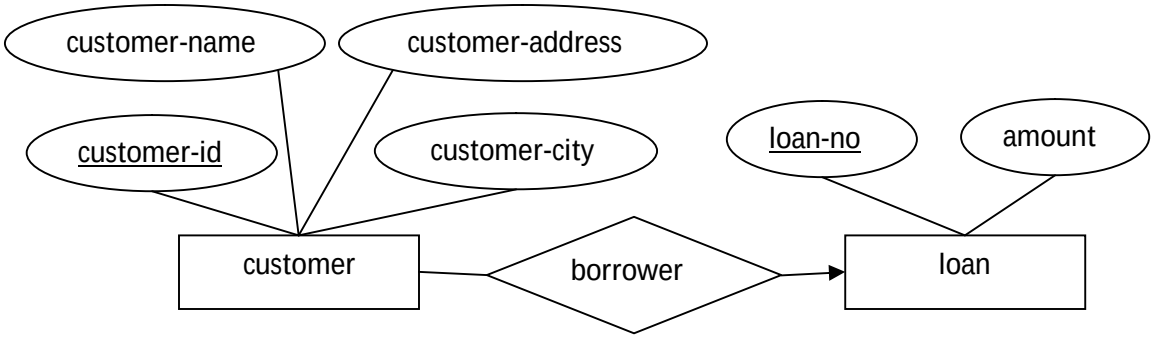


- In the one-to-many relationship a loan is connected with only one customer using borrower and a customer is connected with more than one loans using borrower.

Many-to-one relationship

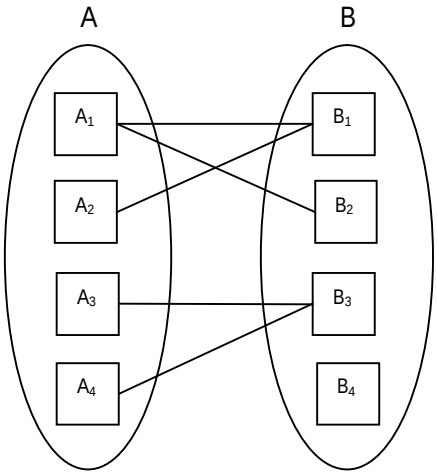


- An entity in A is associated with at most (only) one entity in B and an entity in B is associated with any number (zero or more) of entities in A.

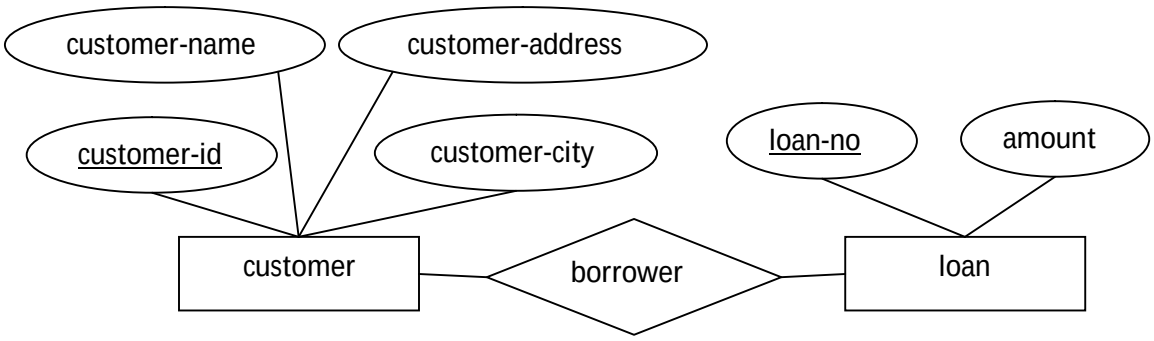


- In a many-to-one relationship a loan is connected with more than one customer using borrower and a customer is connected with only one loan using borrower.

Many-to-many relationship



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



- A customer is connected with more than one loan using borrower and a loan is connected with more than one customer using borrower.

Explain various participation constraints.

Participation constraints

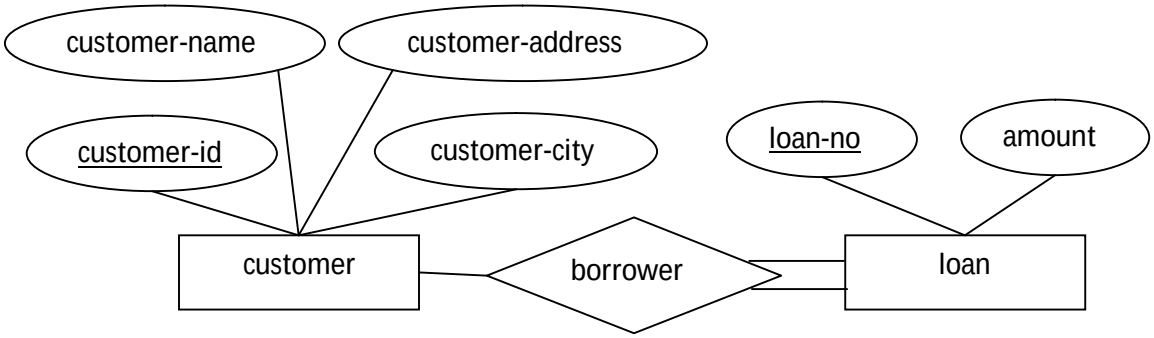
- It specifies the participation of an entity set in a relationship set.
- There are two types participation constraints
  - ✓ Total participation
  - ✓ Partial participation

Total participation

- In total participation every entity in the entity set participates in at least one relationship in the relationship set.
- It specifies that every entity in super class must be member of some of its sub class.
- E.g. participation of loan in borrower is total because every loan must be connected to a customer using borrower.
- It is indicated by double line.

Partial participation

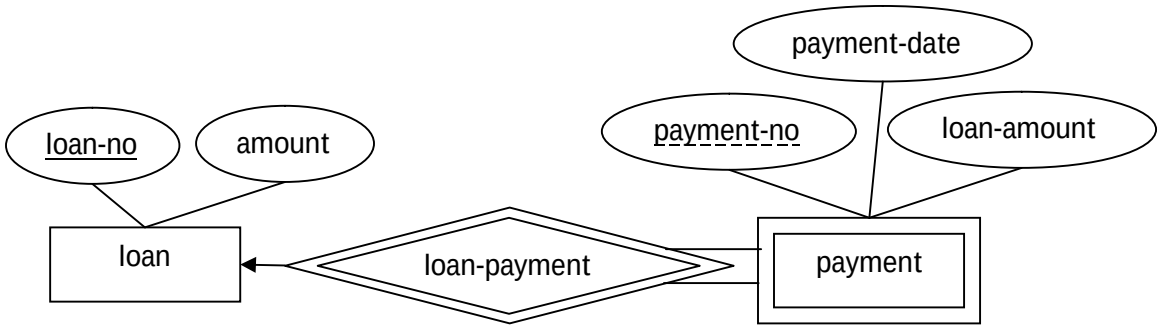
- In partial participation some entities may not participate in any relationship in the relationship set.
- It specifies that an entity may not belong to any sub class.
- E.g. participation of customer in borrower is partial because every customer is not connected to loan using borrower.
- It is indicated by single line.



Explain weak entity set with the help of example.

Weak entity set

- An entity set that does not have a primary key is called weak entity set.
- The existence of a weak entity set depends on the existence of a strong entity set.
- Weak entity set is indicated by double rectangle.
- Weak entity relationship set is indicated by double diamond.
- The discriminator (partial key) of a weak entity set is the set of attributes that distinguishes between all the entities of a weak entity set.
- The primary key of a weak entity set is created by combining the primary key of the strong entity set on which the weak entity set is existence dependent and the weak entity set's discriminator.
- We underline the discriminator attribute of a weak entity set with a dashed line.
- E.g. in below fig. there are two entities loan and payment in which loan is strong entity set and payment is weak entity set.
- Payment entity has payment-no which is discriminator.
- Loan entity has loan-no as primary key.
- So primary key for payment is (loan-no, payment-no).



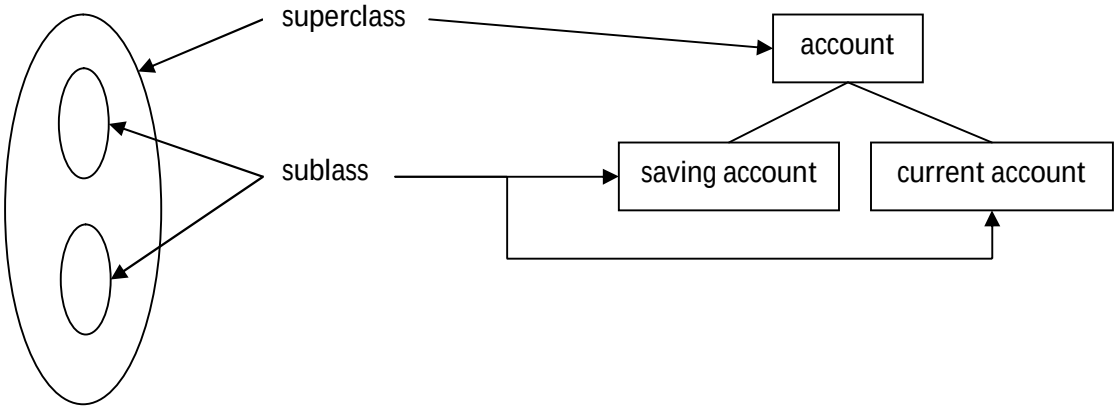
Explain the Superclass and Subclass in E-R diagram with the help of example.

Superclass

- A superclass is an entity from which another entity can be derived.
- A superclass is a generic entity set which has a relationship with one or more subclasses.
- For example, an entity set account has two subsets saving_account and current_account. So an account is superclass.
- Each member of subclass is also a member of superclass. So any saving account or a current account is a member of entity set account.

Subclass

- A subclass is an entity that is derived from another entity.
- A class is a subset of entities in an entity set which has attributes distinct from those in other subset.
- For example, entities of the entity set account are grouped in to two classes saving_account and current_account. So saving_account and current_account are subclasses.



Explain the difference between Specialization and Generalization in E-R diagram.

Specialization	Generalization
It will work in Top-down approach.	It will work in Bottom-up approach
The process of creating sub-groupings within an entity set is called specialization.	The process of creating groupings from various entity sets is called generalization.
Specialization is a process of taking a sub set of higher level entity set to form a lower-level entity set.	Generalization is a process of taking the union of two or more lower-level entity sets to produce a higher-level entity set.
Specialization starts from a single entity set; it creates different low-level entity set using some different features.	Generalization starts from the number of entity sets and creates high-level entity set using some common features.

Explain Specialization and Generalization in E-R diagram with example.

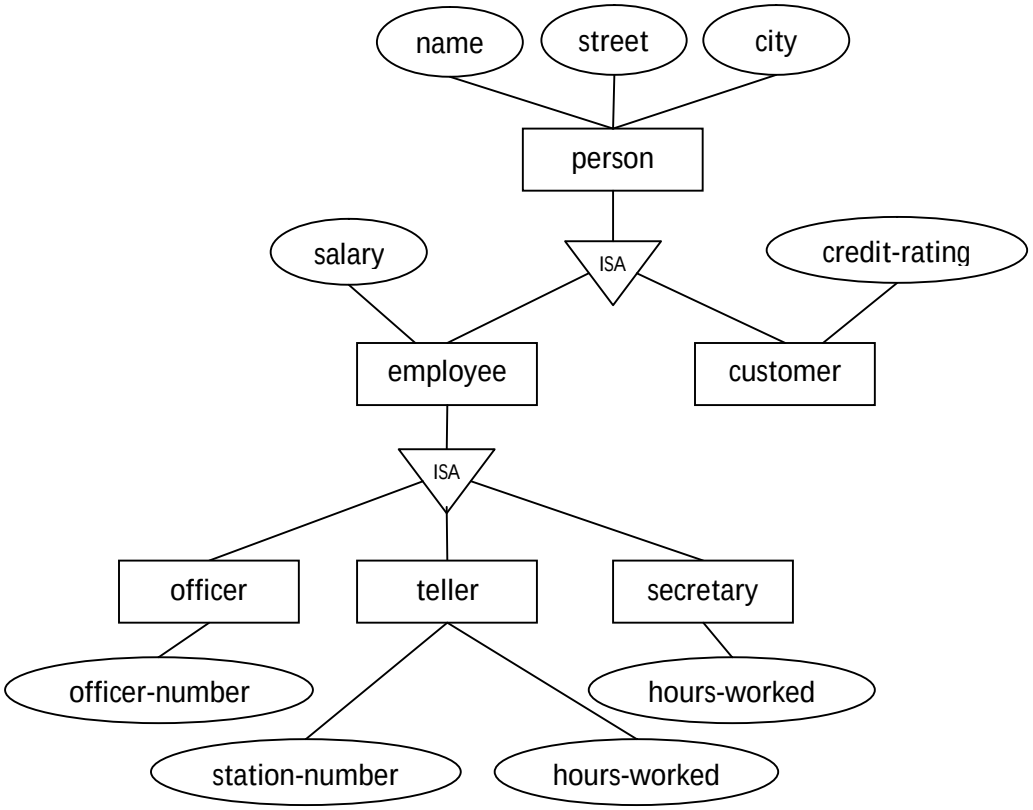
- For all practical purposes, generalization specialization is inversion of each other.

Specialization

- A top-down design process that creates subclasses based on some different characteristics of the entities in the superclass.
- An entity set may include sub groupings of entities that are distinct in some way from other entities in the set.
- For example, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set.
- Consider an entity set person, with attributes name, street and city.
- A person may be further classified as one of the following:
  - ✓ customer
  - ✓ Employee
- For example, customer entities may be described further by the attribute customer-id, credit-rating and employee entities may be described further by the attributes employee-id and salary.
- The process of designating sub groupings within an entity set is called specialization. The specialization of person allows us to distinguish among persons according to whether they are employees or customers.
- Now again, employees may be further classified as one of the following:
  - ✓ officer
  - ✓ teller
  - ✓ secretary
- Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes.
- For example, officer entities may be described further by the attribute office-number, teller entities by the attributes station-number and hours-per-week, and secretary entities by the attribute hours-per-week.
- In terms of an E-R diagram, specialization is depicted by a triangle component labeled ISA.
- The label ISA stands for "is a" and represents, for example, that a customer "is a" person.
- The ISA relationship may also be referred to as a superclass subclass relationship.

Generalization

- A bottom-up design process that combines number of entity sets that have same features into a higher-level entity set.
- The design process proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher level entity set on the basis of common features.
- The database designer may have to first identify a customer entity set with the attributes name, street, city, and customer-id, and an employee entity set with the attributes name, street, city, employee-id, and salary.
- But customer entity set and the employee entity set have some attributes common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher level entity set and one or more lower level entity sets.
- In our example, person is the higher level entity set and customer and employee are lower level entity sets.
- Higher level entity set is called superclass and lower level entity sets is called subclass. So the person entity set is the superclass of two subclasses customer and employee.
- Differences in the two approaches may be characterized by their starting point and overall goal.



Explain types of constraints on specialization and Generalization.

- There are mainly two types of constraints apply to a specialization/generalization:

Disjoint Constraint

- Describes relationship between members of the subclasses and indicates whether member of a superclass can be a member of one, or more than one, subclass.
- It may be disjoint or non-disjoint (overlapping).

Disjoint constraint

- It specifies that the subclasses of the specialization must be disjointed (an entity can be a member of only one of the subclasses of the specialization).
- Specified by 'd' in EER diagram or by writing disjoint.

Non-disjoint (overlapping)

- It specifies that is the same entity may be a member of more than one subclass of the specialization.
- Specified by 'o' in EER diagram or by writing overlapping.

Participation Constraint

- Determines whether every member in super class must participate as a member of a subclass.
- It may be total (mandatory) or partial (optional).

Total (mandatory)

- Total specifies that every entity in the super class must be a member of some subclass in the specialization/ generalization.
- Specified by a double line in EER diagram.

Partial (optional)

- Partial specifies that every entity in the super class not belong to any of the subclasses.
- Specified by a single line in EER diagram.
- Based on these two different kinds of constraints, a specialization or generalization can be one of four types
  - ✓ Total, Disjoint
  - ✓ Total, Overlapping
  - ✓ Partial, Disjoint
  - ✓ Partial, Overlapping.

Explain aggregation in E-R diagram.

- The E-R model cannot express relationships among relationships.
- When would we need such a thing at that time aggregation is used.
- Consider a batabase with information about employees who work on a particular project and use a number of machines doing that work.

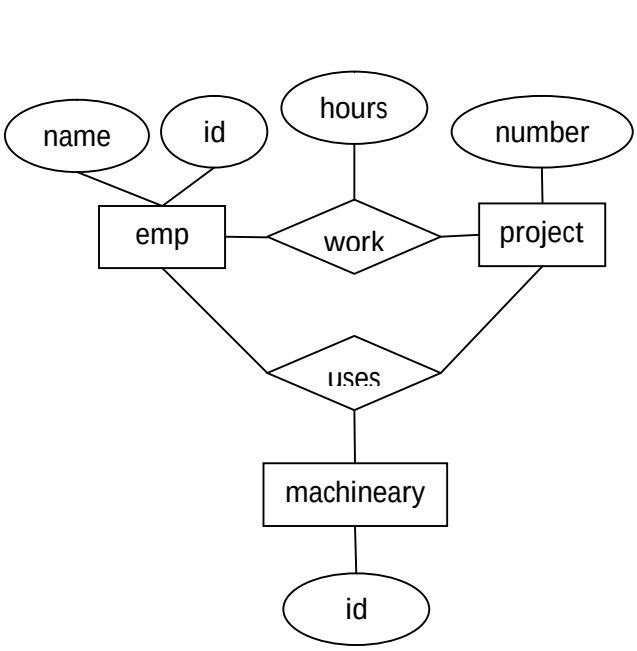


Fig. A

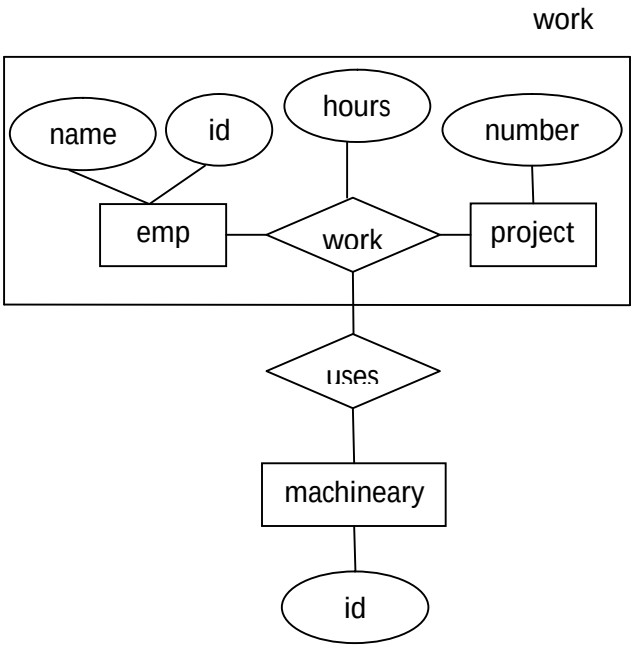


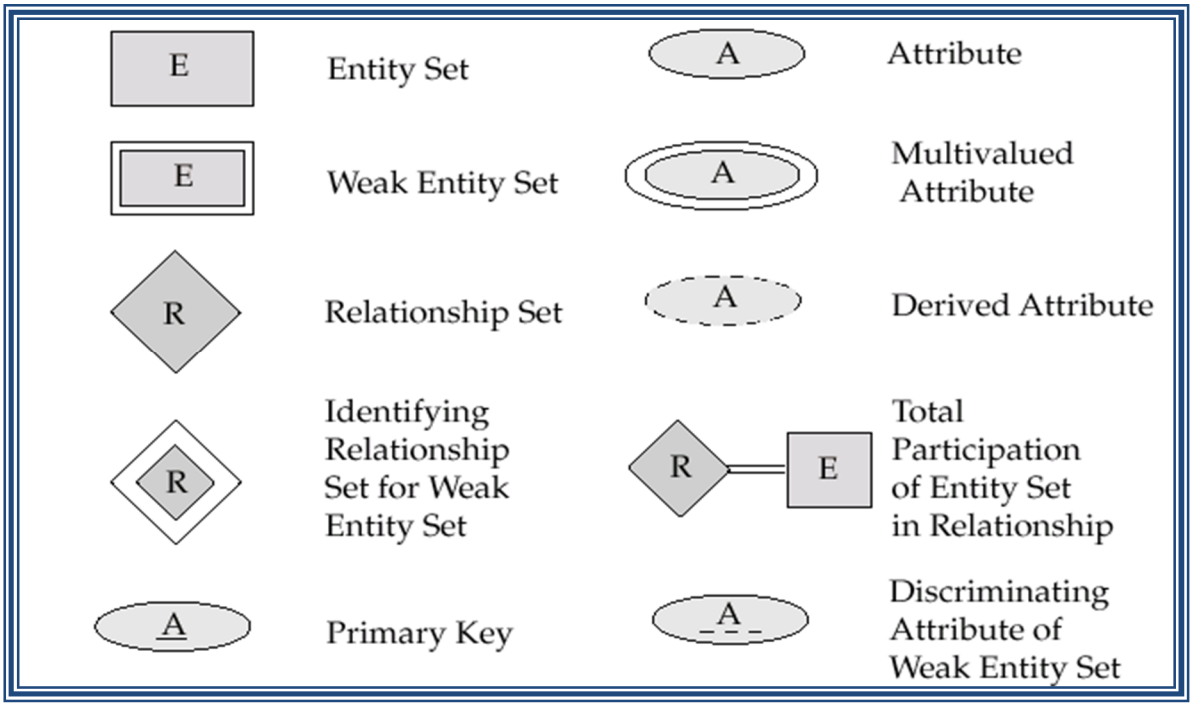
Fig. B

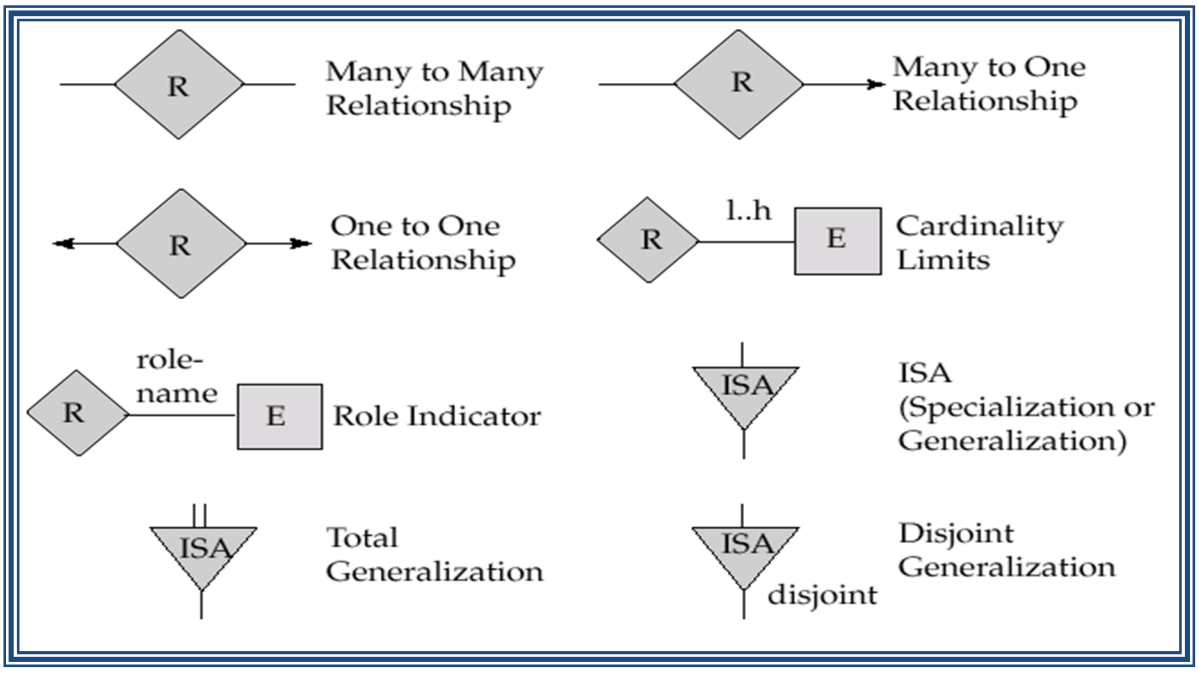
- Relationship sets work and uses could be combined into a single set. We can combine them by using aggregation.
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- For our example, we treat the relationship set work and the entity sets employee and project as a higher-level entity set called work.
- Transforming an E-R diagram with aggregation into tabular form is easy. We create a table for each entity and relationship set as before.
- The table for relationship set uses contains a column for each attribute in the primary key of machinery and work.

What is E-R model (Entity-Relationship) model (diagram) also draw various symbols using in E-R diagram.

E-R model

- Entity-relationship (ER) diagram is a graphical representation of entities and their relationships to each other with their attributes.





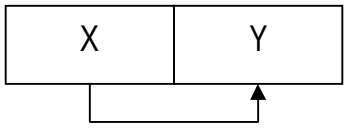
Define functional dependency. Explain different types of FD with the help of example. Explain the use of functional dependency in database design.

Functional Dependency

- Let R be a relation schema having n attributes A1, A2, A3,..., An.
- Let attributes X and Y are two subsets of attributes of relation R.
- If the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component, then, there is a functional dependency from X to Y.
- This is denoted by X → Y.
- It is referred as: Y is functionally dependent on the X, or X functionally determines Y.
- The abbreviation for functional dependency is FD or fd.
- The set of attributes X is called the left hand side of the FD, and Y is called the right hand side.
- The left hand side of the FD is also referred as determinant whereas the right hand side of the FD is referred as dependent.

Diagrammatic representation

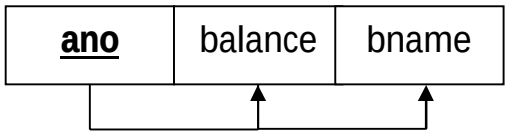
R:



Example

- Consider the relation Account(ano, balance, bname). In this relation ano can determines balance and bname. So, there is a functional dependency from ano to balance and bname.
- This can be denoted by ano → {balance, bname}.

Account:



Types of Functional Dependencies

Full Dependency

- In a relation, the attribute B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- Eg. {Roll_No, Department_Name}→SPI

Partial Dependency

- In a relation, the attribute B is partial functional dependent on A if B is functionally dependent on A as well as on any proper subset of A.
- If there is some attribute that can be removed from A and the dependency still holds.
- Eg. {Enrollment_No, Department_Name}→SPI

Transitive Dependency

- In a relation, if attribute(s) A→B and B→C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

- Eg. Staff_No→Branch_No and Branch_No→Branch_Address

Trivial FD:

- X→Y is trivial FD if Y is a subset of X
- Eg.{Roll_No, Department_Name}→Roll_No

Nontrivial FD

- X→Y is nontrivial FD if Y is not a subset of X
- Eg.. {Roll_No, Department_Name}→Student_Name

Use of functional dependency in database design

- To test relations to see if they are legal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.
- To specify constraints on the set of legal relations

List and explain Armstrong's axioms (inference rules).

- Let A, B, and C is subsets of the attributes of the relation R.

Reflexivity

- If B is a subset of A then A → B

Augmentation

- If A → B then AC → BC

Transitivity

- If A → B and B → C then A → C

Self-determination

- A → A

Decomposition

- If A → BC then A → B and A → C

Union

- If A → B and A → C then A → BC

Composition

- If A → B and C → D then A,C → BD

What is redundant functional dependency? Write an algorithm to find out redundant functional dependency.

Redundant functional dependency

- A FD in the set is redundant, if it can be derived from the other FDs in the set.

Algorithm

Input: Let F be a set of FDs for relation R.
Let f: A → B is a FD to be examined for redundancy.

Steps:

1. F' = F – f # find out new set of FDs by removing f from F
2. T = A # set T = determinant of A → B
3. For each FD: X → Y in F' Do
   If X ⊆ T Then # if X is contained in T
   T = T U Y # add Y to T
   End if

End For
4. If B ⊆ T Then # if B is contained in T
f : A → B is redundant. # given FD f : A → B is redundant. `
End if

Output: Decision whether a given FD f : A → B is redundant or not.

Example

Suppose a relation R is given with attributes A,B, C, D,E.
Also, a set of functional dependencies F is given with following FDs.
F = {A →B, C →D,BD → E, AC → E}

1. Find out whether a FD f: AC → E is redundant or not.

Step-1: F'= { A→ B, C → D, BD → E} #F' = F – f
Step-2:T = AC # set T = determinant of AC → E
Step-3:T = AC + B = ACB # A → B is in F' and A ⊆T
T = ACB+D=ACBD # C→D is in F' and C ⊆ T
T = ACB+E=ACBDE # AC→E is in F' and AC ⊆ T
Step-4: f: AC → E is redundant. #E ⊆ T

2. Find out whether a FD f: BD → E is redundant or not.

Step-1: F'= { A→ B, C → D, AC→ E} #F' = F – f
Step-2:T = BD # set T = determinant of f: BD→ E
Step-3: Nothing can be added to T,
As there is no other FD: X → Y such that X ⊆ T
Step-4: f: BD→ E is not redundant. # E is not contained in T

What is closure of a set of FD? Explain how (Write an algorithm) to find closure of a set of attributes.

Closure of set of FD

- A closure of a set of FDs is a set of all possible FDs that can be derived from a given set of FDs.
- It is also referred as a complete set of FDs.
- If F is used to denote the set of FDs for relation R, then a closure a set of FDs implied by F is denoted by F^+.

Algorithm

- Given a set of attributes a, define the closure of a under F (denoted by a^+) as the set of attributes that are functionally determined by a under F
- Algorithm to compute a^+, the closure of a under *F*

result:= a;
while (changes to result) **do**
for each $\beta \rightarrow \gamma$ **in** F **do**
begin
if $\beta \subseteq$ result **then** result := result $\cup \gamma$
end

Example

- Consider the following relation schema Depositer_Account(cid, ano, acess_date, balance, bname).
- For this relation, a set of functional dependencies F can be given as
  F = { {cid, ano } →access_date andano→ { balance, bname } }
- Find out the closure of F.

Solution

- Determine each set of attributes X that appears as a left-hand side of FD in F.
  { cid, ano } and ano.
- Find out { cid, ano }$^+$
  Step-1 :{ cid, ano }$^+$ = { cid, ano }
  Step-2 :{ cid, ano }$^+$ = { cid, ano, acess_date } # {cid, ano} $\subseteq X^+$
  { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname } # ano$\subseteq X^+$
  Step-3 :{ cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }
- Find out ano$^+$
  Step-1 :ano$^+$ = ano
  Step-2 :ano$^+$ = { ano, balance, bname } # ano$\subseteq X^+$
  Step-3 :ano$^+$ = { ano, balance, bname }
- Combine all such sets of $X^+$to form a closureof F.
  { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }
  ano$^+$ = { ano, balance, bname }

What is decomposition? Explain different types of decomposition.

Decomposition

- Decomposition is the process of breaking down given relation into two or more relations.
- Here, a relation R is replaced by two or more relations in such a way that -
  1. Each new relation contains a subset of the attributes of R, and
  2. Together, they all include all tuples of R.
- Relational database design process starts with a universal relation schema R = {A1, A2, A3,..., An), which includes all the attributes of the database. The universal relation states that every attribute name is unique.
- Using functional dependencies, this universal relation schema is decomposed into a set of relation schemas D = {R1, R2, R3,...,Rm}.
- Now, D becomes the relational database schema and D is referred as decomposition of R.
- Generally, decomposition is used to eliminate the pitfalls of the poor database design during normalization process.
- For example, consider the relation Account_Branch given in figure:

Account_Branch			
Ano	**Balance**	**Bname**	**Baddress**
A01	5000	Vvn	Mota bazaar, VVNagar
A02	6000	Ksad	Chhota bazaar, Karamsad
A03	7000	Anand	Nana bazaar, Anand
A04	8000	Ksad	Chhota bazaar, Karamsad
A05	6000	Vvn	Mota bazaar, VVNagar

- This relation can be devided with two different relations
  1. Account (Ano, Balance, Bname)
  2. Branch (Bname, Baddress)
- These two relations are shown in below figure

Account		
Ano	**Balance**	**Bname**
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	Vvn

Branch	
Bname	**Baddress**
Vvn	Mota bazaar, VVNagar
Ksad	Chhota bazaar, Karamsad
Anand	Nana Bazar, Anand

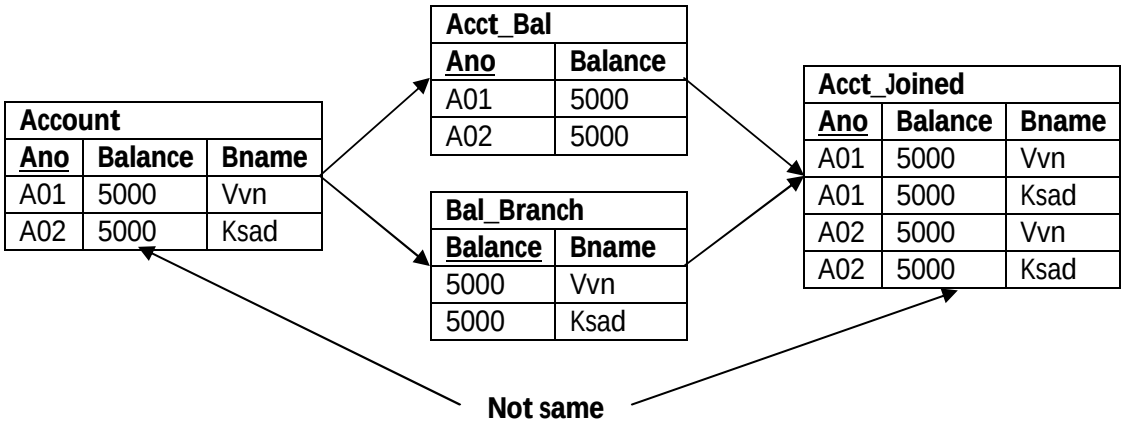
- A decomposition of a relation can be either lossy decomposition or lossless decomposition.
- There are two types of decomposition
  1. lossy decomposition
  2. lossless decomposition (non-loss decomposition)

Lossy Decomposition

- The decomposition of relation R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R.
- This is also referred as lossy-join decomposition.
- The disadvantage of such kind of decomposition is that some information is lost during retrieval of original relation. And so, such kind of decomposition is referred as lossy decomposition.
- From practical point of view, decomposition should not be lossy decomposition.

Example

- A figure shows a relation Account. This relation is decomposed into two relations Acc_Bal and Bal_Branch.
- Now, when these two relations are joined on the common attribute Balance, the resultant relation will look like Acct_Joined. This Acct_Joined relation contains rows in addition to those in original relation Account.
- Here, it is not possible to specify that in which branch account A01 or A02 belongs.
- So, information has been lost by this decomposition and then join operation.



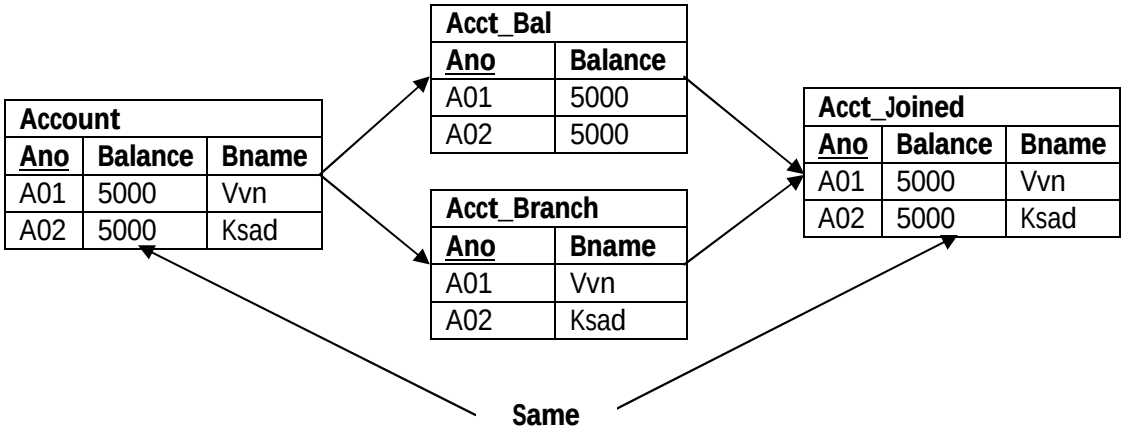
- In other words, decomposition is lossy if decompose into R1 and R2 and again combine (join) R1 and R2 we cannot get original table as R1, over X, where R is an original relation, R1 and R2 are decomposed relations, and X is a common attribute between these two relations.

Lossless (Non-loss) Decomposition

- The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 produces the same relation as in R.
- This is also referred as non-additive (non-loss) decomposition.
- All decompositions must be lossless.

Example

- Again, the same relation Account is decomposed into two relations Acct_Bal and Acct_Branch.
- Now, when these two relations are joined on the common column Ano, the resultant relation will look like Acc_Joined relation. This relation is exactly same as that of original relation Account.
- In other words, all the information of original relation is preserved here.
- In lossless decomposition, no any fake tuples are generated when a natural join is applied to the relations in the decomposition.
- In other words, decomposition is lossy if R = join of R1 and R2, over X, where R is an original relation, R1 an R2 are decomposed relations, and x is a common attribute between these two relations.



What is an anomaly in database design? How it can be solved.

Anomaly in database design:

- Databases are designed to collect data and sort or present it in specific way to the end user.
- Database anomalies, are really just unmatched or missing information caused by limitations or flaws within a given database.
- Entering or deleting information cause issues if the database is limited or has 'bugs'.
- Different types of anomalies in database
    - Insert anomaly
    - Update anomaly
    - Delete anomaly

How anomalies can be solves:

- By using normalization

What is normalization? What is the need of it?

Normalization

- Database normalization is the process of removing redundant data from your tables to improve storage efficiency, data integrity, and scalability.
- In the relational model, methods exist for quantifying how efficient a database is. These classifications are called normal forms (or NF), and there are algorithms for converting a given database between them.
- Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.

Need of Normalization

- Eliminates redundant data
- Reduces chances of data errors
- Reduces disk space
- Improve data integrity, scalability and data consistency.

Explain different types of normal forms with example. OR Explain 1NF, 2NF, 3NF, BCNF, 4NF and 5NF with example.

1NF

- A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only. **OR**
- A relation R is in first normal form (1NF) if and only if it does not contain any composite or multi valued attributes or their combinations.

Example

Cid	Name	Address		Contact_no
		Society	City	
C01	Riya	SaralSoc, Aand		9879898798,55416
C02	Jiya	Birla Gruh, Rajkot		9825098254

- Above relation has four attributes Cid, Name, Address, Contact_no. Here address is composite attribute which is further divided in to sub attributes as Society and City. Another attribute Contact_no is multi valued attribute which can store more than one values. So above relation is not in 1NF.

Problem

- Suppose we want to find all customers for some particular city then it is difficult to retrieve. Reason is city name is combined with society name and stored whole as address.

Solution

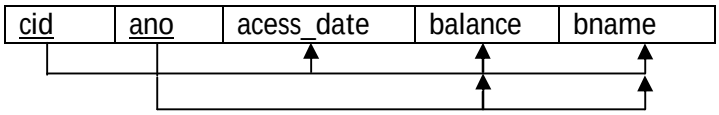
- Insert separate attribute for each sub attribute of composite attribute.
- Insert separate attribute for multi valued attribute and insert only one value on one attribute and other in other attribute.
- So above table can be created as follows.

Cid	Name	Society	City	Contact_no1	Contact_no2
C01	Riya	SaralSoc	Aand	9879898798	55416
C02	Jiya	Birla Gruh	Rajkot	9825098254	

2NF

- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. **OR**
- A relation R is in second normal form (2NF) if and only if it is in 1NF and no any non-key attribute is partially dependent on the primary key.

Example



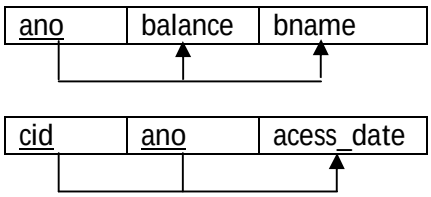
- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {cid,ano}→{acess_date,balance,bname} and
  FD2 ano→{balance,bname}
- We have cid and ano as primary key. As per FD2 balace and bname are only depend on ano not cid. In above table balance and bname are not fully dependent on primary key but these attributes are partial dependent on primary key. So above relation is not in 2NF.

Problem

- For example in case of joint account multiple customers have common accounts. If some account says 'A02' is jointly by two customers says 'C02' and 'C04' then data values for attributes balance and bname will be duplicated in two different tuples of customers 'C02' and 'C04'.

Solution

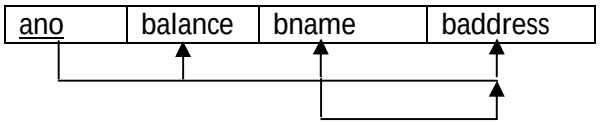
- Decompose relation in such a way that resultant relation does not have any partial FD.
- For this purpose remove partial dependent attribute that violets 2NF from relation. Place them in separate new relation along with the prime attribute on which they are full dependent.
- The primary key of new relation will be the attribute on which it if fully dependent.
- Keep other attribute same as in that table with same primary key.
- So above table can be decomposed as per following.



3NF

- A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that: $A \rightarrow B$ and $B \rightarrow C$.

Example



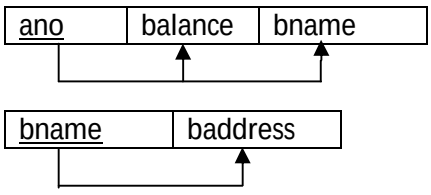
- Above relation has four attributes ano, balance, bname, baddress and two FDS
  FD1 ano→{balance, bname, baddress} and
  FD2 bname→baddress
- So from FD1 and FD2 and using transitivity rule we get ano→baddress.
- So there is transitively dependency from ano to baddress using bname in which baddress is non-prime attribute.
- So there is a non-prime attribute baddress which is transitively dependent on primary key ano.
- So above relation is not in 3NF.

Problem

- Transitively dependency results in data redundancy.
- In this relation branch address will be stored repeatedly from each account of same branch which occupy more space.

Solution

- Decompose relation in such a way that resultant relation does not have any non-prime attribute that are transitively dependent on primary key.
- For this purpose remove transitively dependent attribute that violets 3NF from relation. Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non-prime attribute.
- Keep other attribute same as in that table with same primary key.
- So above table can be decomposed as per following.

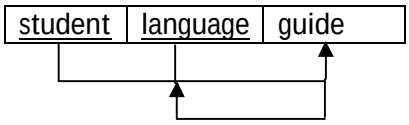


BCNF

- A relation R is in BCNF if and only if it is in 3NF and no any prime attribute is non-transitively dependent on the primary key.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that A→B and B→C.

Example

Student_Project		
Student	**Language**	**Guide**
Mita	JAVA	Patel
Nita	VB	Shah
Sita	JAVA	Jadeja
Gita	VB	Dave
Rita	VB	Shah
Nita	JAVA	Patel
Mita	VB	Dave
Rita	JAVA	Jadeja



- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {student, language}→guide and
  FD2 guide→language
- So from FD1 and FD2 and using transitivity rule we get student→language
- So there is transitively dependency from student to language in which language is prime attribute.
- So there is on prime attribute language which is transitively dependent on primary key student.
- So above relation is not in BCNF.

Problem

- Transitively dependency results in data redundancy.
- In this relation one student have more than one project with different guider then records will be stored repeatedly from each student and language and guides combination which occupies more space.

Solution

- Decompose relation in such a way that resultant relation does not have any prime attribute transitively dependent on primary key.
- For this purpose remove transitively dependent prime attribute that violets BCNF from relation. Place them in separate new relation along with the non prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non prime attribute.
- So above table can be decomposed as per following.

Student	Guide
Mita	Patel
Nita	Shah
Sita	Jadeja
Gita	Dave
Rita	Shah
Nita	Patel
Mita	Dave
Rita	Jadeja

Guide	Language
Patel	JAVA
Shah	VB
Jadeja	JAVA
Dave	VB

4NF

- A table is in the 4NF if it is in BCNF and has no multi-valued dependencies.

Example

- The multi-valued dependency $X \twoheadrightarrow Y$ holds in a relation R if whenever we have two tuples of R that agree (same) in all the attributes of X, then we can swap their Y components and get two new tuples that are also in R.
- Suppose a student can have more than one subject and more than one activity.

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

- Note that all three attributes make up the Primary Key.
- Note that Student_Id can be associated with many subject as well as many activities (multi-valued dependency). Multi-valued dependency leads to modification anomalies.
- Suppose student 100 signs up for skiing. Then we would insert (100, Music, Skiing). This row implies that student 100 skies as Music subject but not as an accounting subject, so in order to keep the data consistent we must add one more row (100, Accounting, Skiing).
- This is an insertion anomaly.
- Here are the tables Normalized

Student_Id	Subject
100	Music
100	Accounting
150	Math

StudentId	Activity
100	Swimming
100	Tennis
150	Jogging

5NF

- A table is in the 5NF it is in 4NF and if for all Join dependency (JD) of (R_1, R_2, R_3, ...,R_m) in R, every R_i is a superkey for R.

- It is also known as Project-join normal form (PJ/NF).

Example

Ranking			
Name	**Code**	**Teaching**	**Research**
Old Town	P01	21	4
New City	C01	23	4

- Candidate keys - NAME or CODE.
- Join dependencies
  JD1 * ((NAME, CODE, TEACHING), (NAME, RESEARCH))
  JD2 * ((NAME, CODE, RESEARCH), (NAME, TEACHING))
  JD3 * ((NAME, CODE, TEACHING), (CODE, RESEARCH))
  JD4 * ((NAME, CODE, RESEARCH), (CODE, TEACHING))
  JD5 * ((NAME, CODE), (NAME, TEACHING), (CODE, RESEARCH))
- all projections in JD1 - to JD5 are super keys for RANKING → 5NF.

Normalize (decompose) following relation into lower to higher normal form.(From 1NF to 4 NF) OR

PLANT	MANAGER	MACHINE	SUPPLIER_NAME	SUPPLIER_CITY
Plant-A	Ravi	Lathe Boiler	Jay industry Abb aplliance	Ahmedabad Surat
Plant-B	Meena	Cutter Boiler CNC	Raj machinery Daksh industry Jay industry	Vadodara Rajkot Ahmedabad

Explain with suitable example, the process of normalization covering from 1NF to 3NF.

1 Normal Form (1NF)

PLANT _ID	PLANT _NAME	MANAGER _ID	MANAGER _NAME	MACHINE _ID	MACHINE _NAME	SUPPLIER _ID	SUPPLIER _NAME	SUPPLIER _CITY
P1	Plant-A	E1	Ravi	M1	Lathe	S1	Jay industry	Ahmedabad
P1	Plant-A	E1	Ravi	M2	Boiler	S2	Abb aplliance	Surat
P2	Plant-B	E2	Meena	M3	Cutter	S3	Raj machinery	Vadodara
P2	Plant-B	E2	Meena	M2	Boiler	S4	Daksh industry	Rajkot
P2	Plant-B	E2	Meena	M4	CNC	S1	Jay industry	Ahmedabad

2 Normal Form (2NF)

Table-1

PLANT_ID	PLANT_NAME	MANAGER_ID	MANAGER_NAME
P1	Plant-A	E1	Ravi
P2	Plant-B	E2	Meena

Table-2

PLANT_ID	MANAGER_ID	MACHINE_NAME
P1	M1	Lathe
P1	M2	Boiler
P2	M3	Cutter
P2	M2	Boiler
P2	M4	CNC

Table-3

MANAGER_ID	SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
E1	S1	Jay industry	Ahmedabad
E1	S2	Abb aplliance	Surat
E2	S3	Raj machinery	Vadodara
E2	S4	Daksh industry	Rajkot
E2	S1	Jay industry	Ahmedabad

3 Normal Form (3NF)

Table-1

PLANT_ID	PLANT_NAME
P1	Plant-A
P2	Plant-B

Table-2

MANAGER_ID	MANAGER_NAME
E1	Ravi
E2	Meena

Table-3

PLANT_ID	MANAGER_ID
P1	E1
P2	E2

Table-4

MACHINE_ID	MANAGER_NAME
M1	Lathe
M2	Boiler
M3	Cutter
M4	CNC

Table-5

PLANT_ID	MANAGER_ID
P1	M1
P1	M2
P2	M3
P2	M2
P2	M4

Table-6

SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
S1	Jay industry	Ahmedabad
S2	Abb aplliance	Surat
S3	Raj machinery	Vadodara
S4	Daksh industry	Rajkot

Table-7

MANAGER_ID	SUPPLIER_ID
E1	S1
E1	S2
E2	S3
E2	S4
E2	S1

4 Normal Form (4NF)

Table-1

PLANT_ID	PLANT_NAME
P1	Plant-A
P2	Plant-B

Table-2

MANAGER_ID	MANAGER_NAME
E1	Ravi
E2	Meena

Table-3

MACHINE_ID	MANAGER_NAME
M1	Lathe
M2	Boiler
M3	Cutter
M4	CNC

Table-4

PLANT_MACHINE_ID	PLANT_ID	MACHINE_ID
PM1	P1	M1
PM2	P1	M2
PM3	P2	M3
PM4	P2	M2
PM5	P2	M4

Table-5

PLANT_MACHINE_ID	MANAGER_ID
PM1	E1
PM2	E1
PM3	E2
PM4	E2
PM5	E2

Table-6

SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
S1	Jay industry	Ahmedabad
S2	Abb aplliance	Surat
S3	Raj machinery	Vadodara
S4	Daksh industry	Rajkot

Table-7

MANAGER_ID	SUPPLIER_ID
E1	S1
E1	S2
E2	S3
E2	S4
E2	S1

Consider a relation R with five attribute A, B, C, D and E having following dependencies:

A -> B, BC -> E and ED -> A

a) List all keys for R.

b) In which normal form table is, justify your answer. **OR**

Consider table R(A,B,C,D,E) with FDs as A->B, BC->E and ED-> A. The table is in which normal form? Justify your answer.

Keys for R are:

{ACD} {BCD} {CDE}

Above relation R is in 3NF because there is no non-prime attributes. That is, every column (attribute) is a part of Super Key, so the right hand side of every FD must be a part of super key. But it's not in BCNF (Or 4NF or 5NF).

What is canonical cover? Consider following set F of functional dependencies on schema R (A, B, C) and compute canonical cover for F. A -> BC, B -> C, A -> B and AB -> C

Canonical cover:

- A canonical cover for F is a set of dependencies Fc such that
  1) F logically implies all dependencies in Fc, and
  2) Fc logically implies all dependencies in F, and
  3) No functional dependency in Fc contains an extraneous attribute, and
  4) Each left side of functional dependency in Fc is unique.

Algorithm to find Canonical cover:

- To compute a canonical cover for F:
  repeat
  Step 1: Use the union rule to replace any dependencies in F
  $\alpha 1 \rightarrow \beta 1$ and $\alpha 1 \rightarrow \beta 2$ with $\alpha 1 \rightarrow \beta 1\ \beta 2$
  Step 2 : Find a functional dependency $\alpha \rightarrow \beta$ with an
  extraneous attribute either in α or in β
  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until F does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Steps to find Canonical Cover:

- Given : R = (A, B, C) and F = {A → BC, B → C, A → B, AB → C}
- Step 1: Combine A → BC and A → B into A → BC
  Set is now {A → BC, B → C, AB → C}
- Step 2: A is extraneous in AB → C
  Check if the result of deleting A from AB → C is implied by the other dependencies
  Yes: in fact, B → C is already present.
  Set is now {A → BC, B → C}
- Step 3: C is extraneous in A → BC
  Check if A → C is logically implied by A → B and the other dependencies
  Yes: using transitivity on A → B and B → C.
  Set is now {A → B and B → C}
  Can use attribute closure of A in more complex cases
- The canonical cover is: A → B, B → C

Given relation R with attributes A, B, C, D,E, F and set of FDs as A-> BC, E-> CF, B->E and CD-> EF. Find out closure {A, B} + of the set of attributes.

Steps to find the closure $\{A,B\}^+$

Step-1: result=AB

Step-2: First loop

result=ABC	# for A → BC, A ⊆ result so result=result ∪ BC
result=ABC	# for E→ CF, E ∉result so result=result
result=ABCE	# for B→E, B ⊆ result so result=result ∪ E
result=ABCE	# for CD→EF, CD ∉result so result=result

result before step2 is AB and after step 2 is ABCE which is different so repeat same as step 2.

Step-3: Second loop

result=ABCE	# for A → BC, A ⊆ result so result=result ∪ BC
result=ABCEF	# for E→ CF, E ⊆ result so result=result ∪ CF
result=ABCEF	# for B→E, B ⊆ result so result=result ∪ E
result=ABCEF	# for CD→EF, CD ∉result so result=result

result before step3 is ABCE and after step 3 is ABCEF which is different so repeat same as step 3.

Step-4: Third loop

result=ABCEF	# for A → BC, A ⊆ result so result=result ∪ BC
result=ABCEF	# for E→ CF, E ⊆ result so result=result ∪ CF
result=ABCEF	# for B→E, B ⊆ result so result=result ∪ E
result=ABCEF	# for CD→EF, CD ∉result so result=result

result before step4 is ABCEf and after step 3 is ABCEF which is same so stop.

So Closure of $\{A, B\}^+$ is {A, B, C, E, F}.

State and explain Heath's Theorem.

Heath's theorem

- A relation R(X, Y, Z) that satisfies a functional dependency X → Y can always be non-loss decomposed into its projections R1=$\prod_{XY}(R)$ and R2=$\prod_{XZ}(R)$.

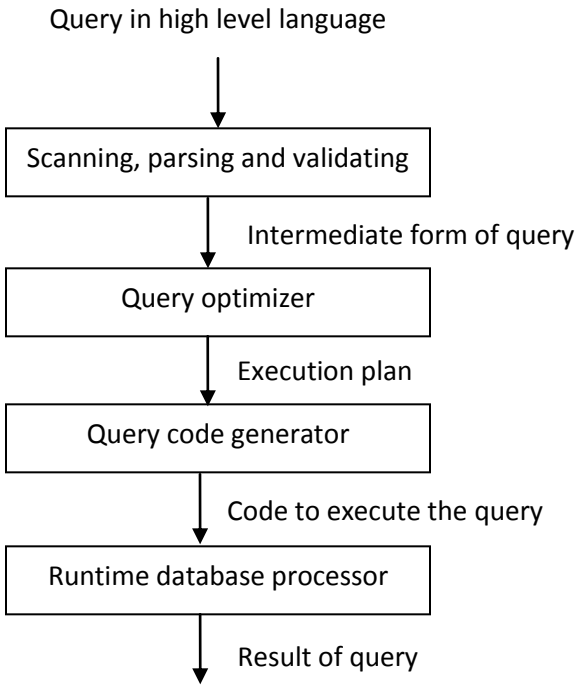
Proof.

- First we show that R ⊆ $\prod_{XY}(R) \bowtie_X \prod_{XZ}(R)$. This actually holds for any relation, does not have to satisfy X → Y.
- Assume r ∈ R. We need to show r ∈ $\prod_{XY}(R) \bowtie_X \prod_{XZ}(R)$.
- Since r ∈ R, r(X,Y) ∈ $\prod_{XY}(R)$ and r(X,Z) ∈ $\prod_{XZ}(R)$.
- Since r(X,Y) and r(X,Z) have the same value for X, their join r(X,Y,Z) = r is in $\prod_{XY}(R) \bowtie_X \prod_{XZ}(R)$.

Explain query processing.

Query processing

- It is a process of transforming a high level query (such as SQL) in to low level language.
- Query processing refers to the range of activities involved in extracting data from a database.



- A query expressed in a high level query language such as SQL must be
  - ✓ Scanned
  - ✓ Parsed
  - ✓ Validated
- The scanner identifies the language tokens such as SQL keywords, attribute names and relation names in the text of the query.
- Parser checks the query syntax to determine whether it is formulated according to the syntax rules of the query language.
- The query must also be validated by checking that all attributes and relation names are valid and semantically meaningful names in the schema of the particular database being queried.
- A query typically has many possible execution strategies and the process of choosing a suitable one for processing a query is known as query optimization.
- The query optimizes module has the task of producing an execution plan and code generator generates the code to execute that plan.
- The runtime database processor has the task of running the query code whether in compiled or interpreted mode, to produce the query result.
- If a runtime error results, an error message is generated by the runtime database processor.
- Query code generator will generate code for query.
- Runtime database processor will select optimal plan and execute query and gives result.

Explain different search algorithm for selection operation. OR Explain linear search and binary search algorithm for selection operation.

- There are two scan algorithms to implement the selection operation:
    1. Linear search
    2. Binary search

Linear search

- In a linear search, the system scans each file block and tests all recordsto see whether they satisfy the selection condition.
- For a selection on a key attribute, the system can terminate the scan if the requires record is found, without looking at the other records of the relation.
- The cost of linear search in terms of number of I/O operations is $\mathbf{b_r}$where$\mathbf{b_r}$is the number of blocks in file.
- Selection on key attribute has an average cost of $\mathbf{b_r/2}$.
- It may be a slower algorithm than any another algorithm.
- This algorithm can be applied to any file regardless of the ordering of file or the availability of indices or the nature of selection operation.

Binary search

- If the file is orderedon attribute and the selection condition is an equality comparisonon the attribute, we can use a binary search to locate the records that satisfy the condition.
- The number of blocksthat need to be examined to find a block containing the required record is $\mathbf{\log(b_r)}$.
- If the selection is on non-key attributemore than one block may contain required records and the cost of reading the extra blocks has to be added to the cost estimate.

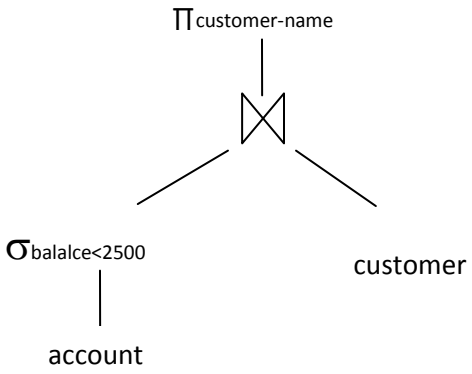
Explain various steps involved in query evaluation. OR Explain query evaluation process. OR Explain evaluation expression process in query optimization.

- There are two methods for the evaluation of expression
    1. Materialization
    2. Pipelining

Materialization

- In this method we start from bottom of the tree and each expression is evaluated one by one in from bottom to top order. Theresult of each expression is materialized(stored) in temporary relation (table) for later use.

$\Pi_{\text{customer-name}}$ ($\sigma_{\text{balalce<2500}}$ (account) $\bowtie$ customer)



- In our example, there is only one such operation, selection operation on account.
- The inputs to the lowest level operation are relations in the database.
- We execute these operations and we store the results in temporary relations.
- We can use these temporary relations to execute the operation at the next level up in the tree, where the inputs now are either temporary relations or relations stored in the database.
- In our example the inputs to join are the customer relation and the temporary relation created by the selection on account.
- The join can now be evaluated, creating another temporary relation.
- By repeating the process, we will finally evaluate the operation at the root of the tree, giving the final result of the expression.
- In our example, we get the final result by executing the projection operation at the root of the tree, using as input the temporary relation created by the join. Evaluation just described is called materialized evaluation, since the results of each intermediate operation are created and then are used for evaluation of the next level operations.
- The cost of a materialized evaluation is not simply the sum of the costs of the operations involved. To compute the cost of evaluating an expression is to add the cost of all the operation as well as the cost of writing intermediate results to disk.
- The disadvantage of this method is that it will create temporary relation (table) and that relation is stored on disk which consumes space on disk.
- It evaluates one operation at a time,starting at the lowestlevel.

Pipelining

- We can reduce the number of temporary files that are produced by combining several relations operations into pipeline operations, in which the results of one operation are passed along to the next operation in the pipeline. Combining operations into a pipeline eliminates the cost reading and writing temporary relations.
- In this method several expression are evaluated as the same time in pipeline by using the result of one operation passed to next without the need to store a temporary relation.

$$\Pi_{\text{customer-name}} \ (\sigma_{\text{balalce<2500}} \ (\text{account}) \bowtie \text{customer})$$

- First it will compute records having balance less than 2500 and then pass this result directly to project without storing that result in any temporary relation (table). And then by using this result it will compute the projections on customer-name.

- It is much cheaper than materialization because in this method no need to store a temporary relation to disk.
- Pipelining is not used in sort, hashjoins.

Explain the method of query optimization. OR Explain query optimization process.

Query optimization

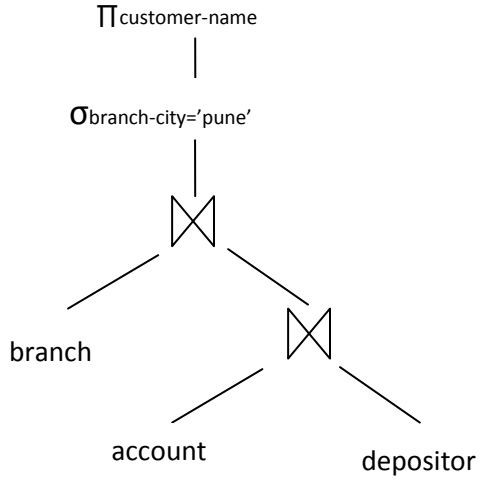
- It is a process of selecting the most efficient query evaluation plan from the available possible plans for processing a given query.
- There are two phases of query optimization
    1. Optimization which occur at the relational algebra level. In this phase the system will find an expression that is equivalent to the given expression but more efficient to execute.
    2. Selecting a detailed strategy for processing the query such as choosing algorithm and specific indices etc.
- For example consider the relational algebra expression for the query
- "Find the name of all customers who have account at any branch located in Pune"

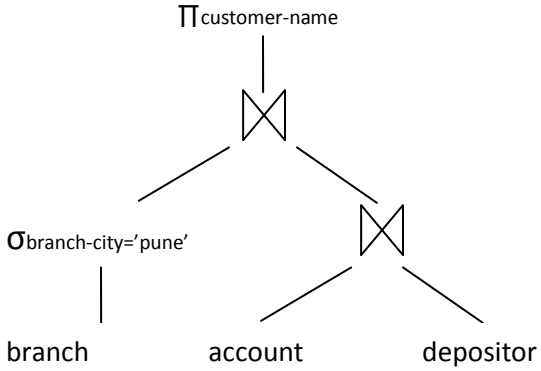
$$\Pi_{\text{customer-name}} (\sigma_{\text{branch-city="pune"}} (\text{branch} \bowtie (\text{account} \bowtie \text{depositor})))$$

- The above query may be written as below

$$\Pi_{\text{customer-name}} (\sigma_{\text{branch-city="pune"}} (\text{branch}) \bowtie (\text{account} \bowtie \text{depositor}))$$

- In the second algebra expression the size of intermediate result is smaller than first because it will only contain the records of pune branch city. Final result of both the expression is same.





- To choose from the different query evaluation plan, the optimizer has to estimate the cost of each evaluation plan.
- Optimizer use statically information about the relation such as relation size and index depth to make a good estimate of the cost of a plan.

Explain transformation of relational expression to equivalent relational expression.

- Two relational algebra expressions are said to be equivalent (same) if on every legal database operation, the two expressions gives the same set of tuples (records). Sequence of records may be different but no of records must be same.

Equivalence rules

- This rule says that expressions of two forms are same.
- We can replace an expression of first form by an expression of the second form.
- The optimizer uses equivalence rule to transform expression into other logically same expression.
- We use

  θ1, θ2, θ3 and so on to denote condition
  L1, L2, L3 and so on to denote list of attributes (columns)
  E1, E2, E3 and so on to denote relational algebra expression.

Rules 1

- Combined selection operation can be divided into sequence of individual selections. This transformation is called cascade of σ.

  $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta 1}(\theta 2\ (E))$

Rules 2

- Selection operations are commutative.

  $\sigma_{\theta 1}(\theta 2\ (E)) = \sigma_{\theta 2}(\theta 1\ (E))$

Rules 3

- If more than one projection operation is used in expression then only the outer projection operation is required. So skip all the other inner projection operation.

  $\prod_{L1}(\prod_{L2}(\ldots(\prod_{Ln}(E))\ldots)) = \prod_{L1}(E)$

Rules 4

- Selection operation can be joined with Cartesian product and theta join.

  $\sigma_{\theta}(E1 \bowtie E2) = E1 \bowtie_{\theta} E2$

  $\sigma_{\theta 1}(E1 \bowtie_{\theta 2} E2) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$

Rules 5

- Theta operations are commutative.

  $E1 \bowtie_{\theta 2} E2 = E2 \bowtie_{\theta 2} E1$

Rules 6

- Natural join operations are associative.

  $(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$

- Theta join operations are associative.

  $(E1 \bowtie_{\theta 1} E2) \bowtie_{\theta 2 \wedge \theta 3} E3 = E1 \bowtie_{\theta 1 \wedge \theta 3} (E2 \bowtie_{\theta 2} E3)$

Rules 7

- The selection operation distribute over theta join operation under the following condition

- ✓ It distribute when all the attributes in the selection condition θ0 involves only the attributes of the one of the expression (says E1) being joined.

  $\sigma_{\theta 0}(E1 \bowtie E2) = (\sigma_{\theta 0}(E1)) \bowtie_{\theta} E2$

- ✓ It distributes when the selection condition θ1involves only the attributes of E1 and θ2 involves only the attributes of E2.

  $\sigma_{\theta 1 \wedge \theta 2}(E1 \bowtie_{\theta} E2) = (\sigma_{\theta 1}(E1) \bowtie_{\theta} (\sigma_{\theta 2}(E2)))$

What is transaction? List and explain ACID property of transaction.

Transaction

- A transaction is a part of program execution that accesses and updates various data items.
- A transaction is a logical unit of work that contains one or more SQL statements.
- A transaction is an atomic unit.
- A database transaction must be atomic, meaning that it must be either entirely completed or aborted.

ACID property

Atomicity

- Either all operations of the transaction are properly reflected in the database or none are. Means either all the operations of a transaction are executed or not a single operation is executed.
- For example consider below transaction to transfer Rs. 50 from account A to account B:
    1. **read**(*A*)
    2. *A* := *A* – 50
    3. **write**(*A*)
    4. **read**(*B*)
    5. *B* := *B* + 50
    6. **write**(*B*)
- In above transaction if Rs. 50 is deducted from account A then it must be added to account B.

Consistency

- Execution of a transaction in isolation preserves the consistency of the database. Means our database must remain in consistent state after execution of any transaction.
- In above example total of A and B must remain same before and after the execution of transaction.

Isolation

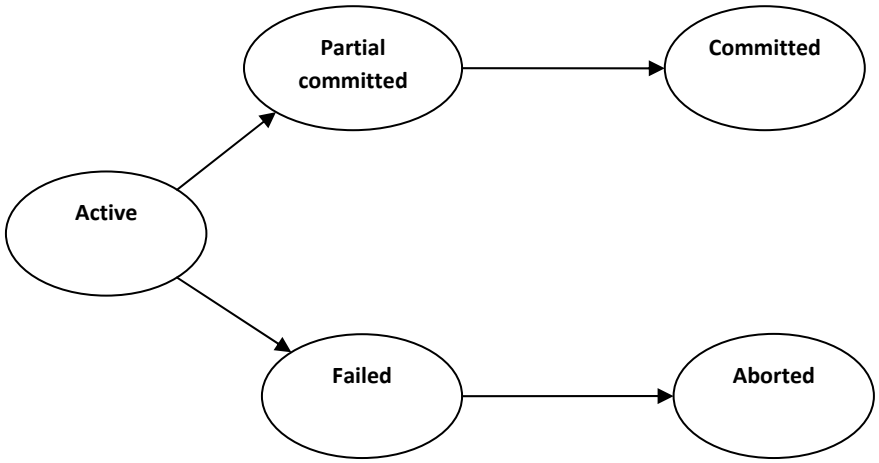
- Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
- In above example once your transaction start from step one its result should not be access by any other transaction until last step (step 6) is completed.

Durability

- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
- Once your transaction completed up to step 6 its result must be stored permanently. It should not be removed if system fails.

Explain different states in transaction processing in database.

- Following are the different states in transaction processing in database
    - ✓ Active
    - ✓ Partial committed
    - ✓ Failed
    - ✓ Aborted
    - ✓ Committed



Active

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

Partially Committed

- This is the state after the final statement of the transaction is executed.

Failed

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

Aborted

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

Committed

- The state after successful completion of the transaction.

Explain following terms.

Schedule

- A schedule is the chronological order in which instructions are executed in a system.
- A schedule for a set of transaction must consist of all the instruction of those transactions and must preserve the order in which the instructions appear in each individual transaction.
- Example of schedule (Schedule 1)

T1	T2
read(A)	
A:=A-50	
write(A)	
read(B)	
B:= B+ 50	
write(B)	
	read(A)
	temp: A * 0.1
	A: A-temp
	write (A)
	read(B)
	B:=B +temp
	write(B)

Serial schedule

- Schedule that does not interleave the actions of different transactions.
- In schedule 1 the all the instructions of T1 are grouped and run together. Then all the instructions of T2 are grouped and run together. Means schedule 2 will not start until all the instructions of schedule 1 are complete. This type of schedules is called serial.

Equivalent schedules

- Two schedule are equivalent schedule if the effect of executing the first schedule is identical (same) to the effect of executing the second schedule.
- We can also says that two schedule are equivalent schedule if the output of executing the first schedule is identical (same) to the output of executing the second schedule.

Serializable schedule

- A schedule that is equivalent (in its outcome) to a serial schedule has the serializability property.
- Example of serializable schedule

Schedule 1		
T1	**T2**	**T3**
read(X)		
write(X)		
	read(Y)	
	write(Y)	
		read(Z)
		write(Z)

Schedule 2		
T1	**T2**	**T3**
read(X)		
	read(Y)	
		read(Z)
write(X)		
	write(Y)	
		write(Z)

- In above example there are two schedules as schedule 1 and schedule 2.
- In schedule 1 and schedule 2 the order in which the actions of transaction are executed is not the same but whatever the result we get is same. So this is known as serializabilty of transaction.

Explain conflict serializability with example.

Conflict serializability

- Instructions I_i and I_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.
  1. I_i = read(Q), I_j = read(Q). I_i and I_j don't conflict.
  2. I_i = read(Q), I_j = write(Q). I_i and I_j conflict.
  3. I_i = write(Q), I_j = read(Q). I_i and I_j conflict.
  4. I_i = write(Q), I_j = write(Q). I_i and I_j conflict.
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
- If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are conflict equivalent.
- We say that a schedule *S* is conflict serializable if it is conflict equivalent to a serial schedule.

Example

- Schedule S can be transformed into Schedule S' by swapping of non-conflicting series of instructions.
- Therefore Schedule S is conflict serializable.

Schedule S	
T1	**T2**
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule S'	
T1	**T2**
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

- Instruction Ii of transaction T1 and Ij of transaction T2 conflict if both of these instruction access same data A and one of these two instructions performs write operation on that data (A).
- In above example the write(A) instruction of transaction T1 conflict with read(A) instruction of transaction T2 because both the instructions access same data A. But write(A) instruction of transaction T2 is not conflict with read(B) instruction of transaction T1 because both the instructions access different data. Transaction T2 performs write operation in A and transaction T1 is reading B.
- So in above example in schedule S two instructions read(A) and write(A) of transaction T2 and two instructions read(B) and write(B) of transaction T1 are interchanged and we get schedule S'.
- Therefore Schedule S is conflict serializable.

Schedule S''	
T3	**T4**
read(Q)	
	write(Q)
write(Q)	

- We are unable to swap instructions in the above schedule S'' to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >.
- So above schedule S'' is not conflict serializable.

Explain view serializability with example.

View serializability

- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are view equivalent if the following three conditions are met, for each data item *Q,*
    1. If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
    2. If in schedule S transaction T_i executes read(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same write(Q) operation of transaction T_j .
    3. The transaction (if any) that performs the final write(Q) operation in schedule S must also perform the final write(Q) operation in schedule S'.
- A schedule *S* is view serializable if it is view equivalent to a serial schedule.

- Every conflict serializable schedule is also view serializable but every view serializable is not conflict serializable.
- Below is a schedule which is view serializable but not conflict serializable.

Schedule S		
T3	**T4**	**T6**
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Above schedule is view serializable but not conflict serializable because all the transactions can use same data item (Q) and all the operations are conflict with each other due to one operation is write on data item (Q) and that's why we cannot interchange any non conflict operation of any transaction.

Explain two phase commit protocol.

Two phase commit protocol

- The two phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction.
- The two phase commit protocol ensures that all participants perform the same action (either to commit or to roll back a transaction).
- The two phase commit strategy is designed to ensure that either all the databases are updated or none of them, so that the databases remain synchronized.
- In two phase commit protocol there is one node which is act as a coordinator and all other participating node are known as cohorts or participant.
- Coordinator – the component that coordinates with all the participants.
- Cohorts (Participants) – each individual node except coordinator are participant.
- As the name suggests, the two phase commit protocol involves two phases.
    1. The first phase is Commit Request phase OR phase 1
    2. The second phase is Commit phase OR phase 2

Commit Request Phase (Obtaining Decision)

- To commit the transaction, the coordinator sends a request asking for **"**ready for commit**"** to each cohort.
- The coordinator waits until it has received a reply from all cohorts to **"**vote**"** on the request.
- Each participant votes by sending a message back to the coordinator as follows:
    - ✓ It vote YES if it is prepared to commit
    - ✓ It may vote NO for any reason if it cannot prepare the transaction due to a local failure.
    - ✓ It may delay in voting because cohort was busy with other work.

Commit Phase (Performing Decision)

- If the coordinator receives YES response from all cohorts, it decides to commit. The transaction is now officially committed. Otherwise, it either receives a NO response or gives up waiting for some cohort, so it decides to abort.
- The coordinator sends its decision to all participants (i.e. COMMIT or ABORT).
- Participants acknowledge receipt of commit or about by replying DONE.

Explain Log based recovery method.

Log based recovery

- The most widely used structure for recording database modification is the log.
- The log is a sequence of log records, recording all the update activities in the database.
- In short Transaction log is a journal or simply a data file, which contains history of all transaction performed and maintained on stable storage.
- Since the log contains a complete record of all database activity, the volume of data stored in the log may become unreasonable large.
- For log records to be useful for recovery from system and disk failures, the log must reside on stable storage.
- Log contains
    1. Start of transaction
    2. Transaction-id
    3. Record-id
    4. Type of operation (insert, update, delete)
    5. Old value, new value
    6. End of transaction that is committing or aborted.
- All such files are maintained by DBMS itself. Normally these are sequential files.
- Recovery has two factors Rollback (Und) and Roll forward (Redo).
- When transaction T_i starts, it registers itself by writing a <T_i start>log record
- Before T_i executes write(X), a log record <T_i, X, V_1, V_2> is written, where V_1 is the value of X before the write, and V_2 is the value to be written to X.
    - ✓ Log record notes that T_i has performed a write on data item X_j
    - ✓ X_j had value V_1 before the write, and will have value V_2 after the write.
- When T_i finishes it last statement, the log record <T_i commit> is written.
- Two approaches are used in log based recovery
    1. Deferred database modification
    2. Immediate database modification

Log based Recovery Techniques

- Once a failure occurs, DBMS retrieves the database using the back-up of database and transaction log. Various log based recovery techniques used by DBMS are as per below:
    1. **Deferred Database Modification**
    2. **Immediate Database Modification**
- Both of the techniques use transaction logs. These techniques are explained in following sub-sections.

Deferred Database Modification

Concept

- Updates (changes) to the database are deferred (or postponed) until the transaction commits.
- During the execution of transaction, updates are recorded only in the transaction log and in buffers. After the transaction commits, these updates are recorded in the database.

When failure occurs

- If transaction has not committed, then it has not affected the database. And so, no need to do any undoing operations. Just restart the transaction.
- If transaction has committed, then, still, it may not have modified the database. And so, redo the updates of the transaction.

Transaction Log

- In this technique, transaction log is used in following ways:
- Transaction T starts by writing <T start> to the log.
- Any update is recorded as <T, X, V>, where V indicates new value for data item X. Here, no need to preserve old value of the changed data item. Also, V is not written to the X in database, but it is deferred.
- Transaction T commits by writing <T commit> to the log. Once this is entered in log, actual updates are recorded to the database.
- If a transaction T aborts, the transaction log record is ignored, and no any updates are recorded to the database.

Example

- Consider the following two transactions, T_0 and T_1 given in figure, where T_0 executes before T_1. Also consider that initial values for A, B and C are 500, 600 and 700 respectively.

Transaction – T_0	Transaction – T_1
Read (A) A =A -100 Write (A) Read (B) B =B+ 100 Write (B)	Read (C) C=C-200 Write (C)

- The following figure shows the transaction log for above two transactions at three different instances of time.

Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start> < T_0, A, 400> < T_0, B, 700>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> <T_1 start> <T_1, C, 500>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> < T_1 start> < T_1, C, 500> < T_1 commit>

- If failure occurs in case of
  1. No any REDO actions are required.
  2. As Transaction T_0 has already committed, it must be redone.
  3. As Transactions T_0 and T_1 have already committed, they must be redone.

Immediate Database Modification

Concept

- Updates (changes) to the database are applied immediately as they occur without waiting to reach to the commit point.
- Also, these updates are recorded in the transaction log.
- It is possible here that updates of the uncommitted transaction are also written to the database. And, other transactions can access these updated values.

When failure occurs

- If transaction has not committed, then it may have modified the database. And so, undo the updates of the transaction.

- If transaction has committed, then, still, it may not have modified the database. And so, redo the updates of the transaction.

Transaction Log

- In this technique, transaction log is used in following ways:
- Transaction T starts by writing <T start> to the log.
- Any update is recorded as <T, X, V_{old}, V_{new} > where V_{old} indicates the original value of data item X and V_{old} indicates new value for X. Here, as undo operation is required, it requires to preserve old value of the changed data item.
- Transaction T commits by writing <T commit> to the log.
- If a transaction T aborts, the transaction log record is consulted, and required undo operations are performed.

Example

- Again, consider the two transactions, T_0 and T_1, given in figure, where T_0 executes before T_1.
- Also consider that initial values for A, B and C are 500, 600 and 700 respectively.
- The following figure shows the transaction log for above two transactions at three different instances of time. Note that, here, transaction log contains original values also along with new updated values for data items.
- If failure occurs in case of -
- Undo the transaction T0 as it has not committed, and restore A and B to 500 and 600 respectively.
- Undo the transaction T1, restore C to 700; and, Redo the Transaction T0 set A and B to 400 and 700 respectively.
- Redo the Transaction T0 and Transaction T0; and, set A and B to 400 and 700 respectively, while set C to 500.

Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700>	< T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700> < T_0 commit> <T_1 start> <T_1, C, 700, 500>	< T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700> < T_0 commit> < T_1 start> < T_1, C, 700, 500> < T_1 commit>

Checkpoints

Problems with Deferred & Immediate Updates

- Searching the entire log is time-consuming.
- It is possible to redo transactions that have already been stored their updates to the database.

Checkpoint

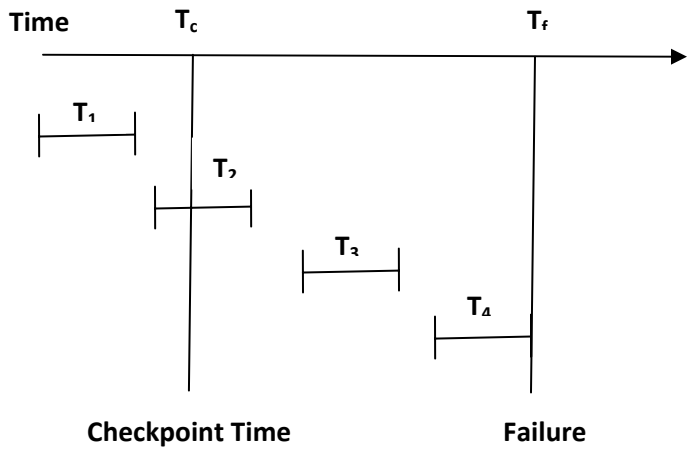
- A point of synchronization between database and transaction log file.
- Specifies that any operations executed before this point are done correctly and stored safely.
- At this point, all the buffers are force-fully written to the secondary storage.
- Checkpoints are scheduled at predetermined time intervals
- Used to limit -
    1. The size of transaction log file
    2. Amount of searching, and
    3. Subsequent processing that is required to carry out on the transaction log file.

When failure occurs

- Find out the nearest checkpoint.
- If transaction has already committed before this checkpoint, ignore it.
- If transaction is active at this point or after this point and has committed before failure, redo that transaction.
- If transaction is active at this point or after this point and has not committed, undo that transaction.

Example

- Consider the transactions given in following figure. Here, T_c indicates checkpoint, while T_f indicates failure time.
- Here, at failure time -
    1. Ignore the transaction T_1 as it has already been committed before checkpoint.
    2. Redo transaction T_2 and T_3 as they are active at/after checkpoint, but have committed before failure.
    3. Undo transaction T_4 as it is active after checkpoint and has not committed.



Shadow Paging

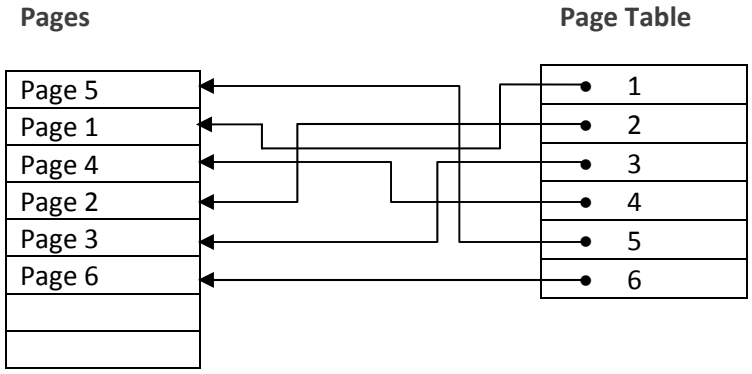
Concept

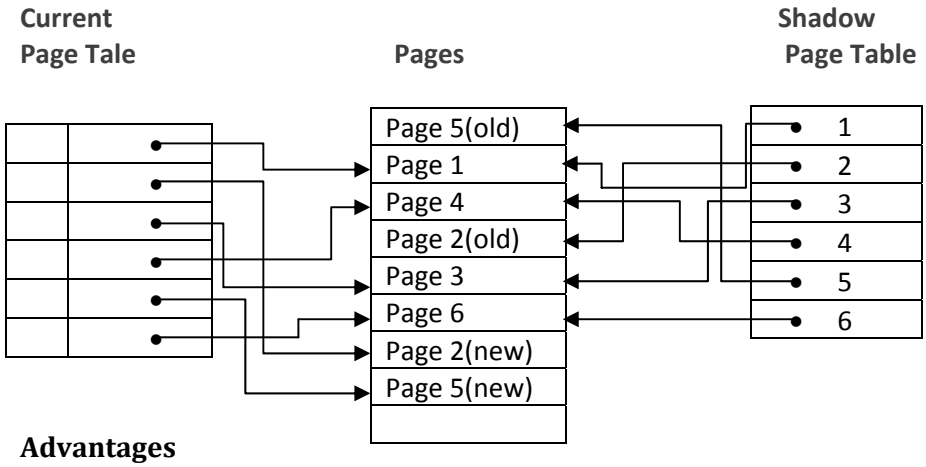
- Shadow paging is an alternative to transaction-log based recovery techniques.
- Here, database is considered as made up of fixed size disk blocks, called pages. These pages are mapped to physical storage using a table, called page table.
- The page table is indexed by a page number of the database. The information about physical pages, in which database pages are stored, is kept in this page table.
- This technique is similar to paging technique used by Operating Systems to allocate memory, particularly to manage virtual memory.
- The following figure depicts the concept of shadow paging.

Execution of Transaction

- During the execution of the transaction, two page tables are maintained.
    1. **Current Page Table**: Used to access data items during transaction execution.
    2. **Shadow Page Table**: Original page table, and will not get modified during transaction execution.

- Whenever any page is about to be written for the first time
    1. A copy of this page is made onto an free page,
    2. The current page table is made to point to the copy,
    3. The update is made on this copy.
- At the start of the transaction, both tables are same and point· to same pages.
- The shadow page table is never changed, and is used to restore the database in case of any failure occurs. However, current page table entries may change during transaction execution, as it is used to record all updates made to the database.
- When the transaction completes, the current page table becomes shadow page table. At this time, it is considered that the transaction has committed.
- The following figure explains working of this technique.
- As shown in this figure, two pages - page 2 & 5 - are affected by a transaction and copied to new physical pages. The current page table points to these pages.
- The shadow page table continues to point to old pages which are not changed by the transaction. So, this table and pages are used for undoing the transaction.





Advantages

- No overhead of maintaining transaction log.
- Recovery is quite faster, as there is no any redo or undo operations required.

Disadvantages

- Copying the entire· page table is very expensive.
- Data are scatters, or fragmented.
- After each transaction, free pages need to be collected by garbage collector. Difficult to extend this technique to allow concurrent transactions.

What is data security?

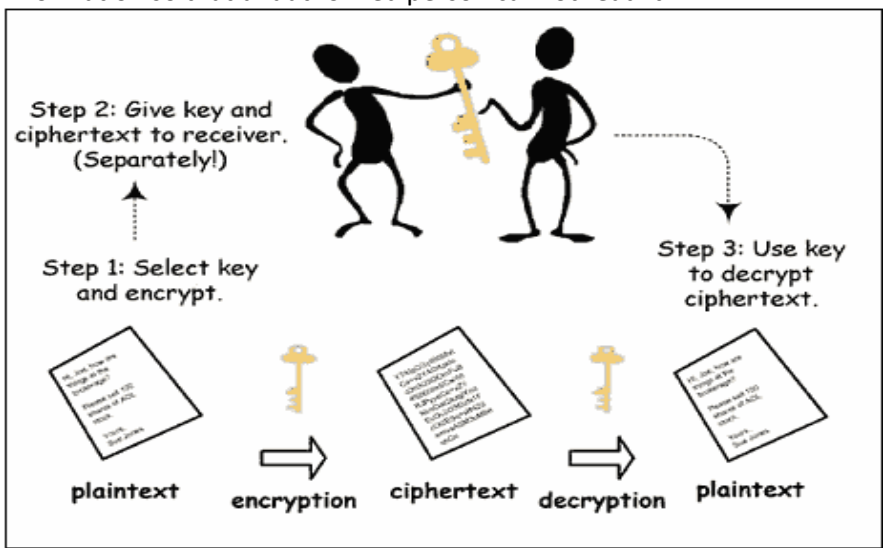
Data security

- Data security is the protection of the data from unauthorized users.
- Only the authorized users are allowed to access the data.
- Most of the users are allowed to access a part of database i.e., the data that is related to them or related to their department.
- Mostly, the DBA or head of department can access all the data in the database.
- Some users may be permitted only to retrieve data, whereas others are allowed to retrieve as well as to update data.
- The database access is controlled by the DBA.
- He/she creates the accounts of users and gives rights to access the database.
- Users or group of users are given usernames protected by passwords.
- The user enters his/her account number (or user name) and password to access the data from database.
- For example, if you have an account in the "yahoo.com" (a popular website), then you have to give your correct username and password to access your account of e-mail. Similarly, when you insert your ATM card into the auto teller machine (ATM) in a ATM machine, the machine reads your ID number printed on the card and then asks you to enter your pin code (or password). In this way you can access your account.

Explain data encryption in detail (brief). OR Write short note on data encryption OR Describe data encryption.

Data encryption

- Encryption is a technique of encoding data, so that only authorized user can understand (read) it.
- The data encryption technique converts readable data into unreadable data by using some information so that unauthorized person cannot read it



Data encryption process

- In above figure sender having data that he/she wants to send is known as plaintext.
- In first step sender will encrypt data (plain text) using encryption algorithm and some key.
- After encryption the plaintext becomes ciphertext.
- This ciphertext is not able to read by any unauthorized person.
- This ciphertext is send to receiver.
- The sender will send that key separately to receiver.
- Once receiver receives this ciphertext he/she will decrypt it using that key send by sender and decryption algorithm.
- After applying decryption algorithm and key receiver will get original data (plaintext) that is sended by sender.
- This technique is used to protect data when there a chance of data theft.
- In such situation if encrypted is theft then it cannot be used (read) directly without knowing the encryption technique and key.
- There are two different method of data encryption
    1. Symmetric encryption
    2. Public key encryption

Explain types of access control methods of data security OR Explain discretionary access control and mandatory access control of data security.

- There are two different methods of data access control:-
    1. Discretionary access control
    2. Mandatory access control

Discretionary access control

- In this method, user is given access rights (privileges) to access database object or data items such as table or view.
- This method is based on the concept of access rights and mechanisms for giving rights to user.
- In this method a single user can have different rights on different data items, as well as different user can have different rights on the same data item.
- SQL support discretionary access control through the GRANT and REVOKE commands.

GRANT

- This command gives rights to user for a data items.

  Syntax:-

  GRANT privilege ON object TO user [WITH GRANT OPTION]

REVOKE

- This command takes back rights from user for a data items.

  Syntax:-

  REVOKE privilege ON object FROM user {RESTRICT/CASCADE}
- Privileges are various operations that we will perform on table or view E.g. INSERT, UPDATE, DELETE, SELECT etc.
- Object is table or view or any data item.
- User is the name of user.

- With grant option is used if user wants to pass the rights to other user. Means if a user get a rights with the grant option then he/she can give this rights to another user.
- When user executes a REVOKE command with the cascade option then it will take back given rights from all the users who get those rights from this user.

Mandatory access control

- In this method individual user cannot get rights. But all the users as well as all the objects are classified into different categories. Each user is assigned a clearance level and each object is assigned a security level.
- A user can access object of particular security level only if he has proper clearance level.
- The DBMS determines whether a given user can read or write a given object based on some rules.
- This rule contains security level of object and clearance level of the user.
- This rule makes sure that sensitive data can never be passed to a user without necessary clearance.

Components

- The commonly used mandatory access control technique for multi-level security uses four components as given below
    1. **Subjects:-**Such as users, accounts, programs etc.
    2. **Objects:-**Such as relation (table), tuples (records), attribute (column), view etc.
    3. **Clearance level:-**Such as top secret (TS), secret (S), confidential (C), Unclassified (U). Each subject is classified into one of these four classes.
    4. **Security level: -**Such as top secret (TS), secret (S), confidential (C), Unclassified (U). Each object is classified into one of these four classes.
- In the above system TS>S>C>U, where TS>S means class TS data is more sensitive than class S data.

Rules:-

- A user can access data by following two rules
    1. Security property:-
        Subject S will read object O if class (S)>=class (O)
    2. Star (*) security property:-
        Subject S will writeobject O if class (S) <=class (O)

Write short note on stored procedure.

- A stored procedure (proc) is a group of PL/SQL statements that performs specific task.
- A procedure has two parts, header and body.
- The header consists of the name of the procedure and the parameters passed to the procedure.
- The body consists of declaration section, execution section and exception section.
- A procedure may or may not return any value. A procedure may return more than one value.

General Syntax to create a procedure

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
        Declaration section
BEGIN
        Execution section
EXCEPTION
        Exception section
END;
```

Explanation

Create:-It will create a procedure.
Replace:- It will re-create a procedure if it already exists.
We can pass parameters to the procedures in three ways.

1. IN-parameters: - These types of parameters are used to send values to stored procedures.
2. OUT-parameters: - These types of parameters are used to get values from stored procedures. This is similar to a return type in functions but procedure can return values for more than one parameters.
3. IN OUT-parameters: - This type of parameter allows us to pass values into a procedure and get output values from the procedure.

IS indicates the beginning of the body of the procedure. The code between IS and BEGIN forms the Declaration section.
Begin:-It contains the executable statement.
Exception:- It contains exception handling part. This section is optional.
End:- It will end the procedure.
The syntax within the brackets [ ] indicates that they are optional.
By using CREATE OR REPLACE together the procedure is created if it does not exist and if it exists then it is replaced with the current code.

How to execute a Stored Procedure?

- There are two ways to execute a procedure.
    1) From the SQL prompt.
       Syntax: EXECUTE [or EXEC] procedure_name (parameter);
    2) Within another procedure – simply use the procedure name.
       Syntax: procedure_name (parameter);

Example 1 (Using IN)

```
CREATE OR REPLACE PROCEDURE get_studentname_by_id (id IN NUMBER)
IS
BEGIN
        SELECT studentname
        FROM stu_tbl
```

```
        WHERE studentID = id;
    END;
```

Execute:- EXECUTE get_studentname_by_id(10); **OR**
get_studentname_by_id(10);

Explanation:- Above procedure gives the name of student whose id is 10.

Example 2 (Using OUT)

```
CREATE OR REPLACE PROCEDURE multiplication @mult1 int, @mult2 int, @result int OUTPUT
as
select @result = @mult1 * @mult2
```

Execute:- EXECUTE multiplication (5,6); **OR**
multiplication (5,6);

Explanation:- Above procedure gives the multiplication of 4 and 6 that is equal to 30.

Advantages of procedure

Security:- We can improve security by giving rights to selected persons only.

Faster Execution:- It is precompiled so compilation of procedure is not required every time you call it.

Sharing of code:- Once procedure is created and stored, it can be used by more than one user.

Productivity:- Code written in procedure is shared by all programmers. This eliminates redundant coding by multiple programmers so overall improvement in productivity.

Explain database trigger with example. OR Write short note on database triggers in PL/SQL.

- A trigger is a PL/SQL block structure which is triggered automatically when DML statements like Insert, Delete, and Update is executed on a table.
- There are two types of triggers based on the level it is triggered.
    1. Row level trigger - An event is triggered when any row of the table is changed irrespective of insert, update or delete statement.
    2. Statement level trigger - An event is triggered when particular SQL statement is executed, e.g. trigger on insert command.
- We cannot pass parameters into triggers like stored procedure.
- We cannot do transaction control (commit … rollback) in trigger.
- Triggers are normally slow.
- When triggers can be used,
    1. Based on change in one table, we want to update other table.
    2. Automatically update derived columns whose values change based on other columns.
    3. Logging.
    4. Enforce business rules.

Syntax of Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
  [BEFORE / AFTER]
  [INSERT / UPDATE / DELETE [of columnname]]
 ON table_name
```

```
  [REFERENCING [OLD AS old, NEW AS new]]
  [FOR EACH ROW [WHEN condition]]
DECLARE
  Declaration section
BEGIN
  Executable statements
END;
```

CREATE [OR REPLACE] TRIGGER trigger_name:- This clause creates a trigger with the given name or overwrites an existing trigger.

[BEFORE | AFTER]:- This clause indicates at what time the trigger should be fired. Before updating the table or after updating a table.

[INSERT / UPDATE / DELETE]:- This clause determines on which kind of statement the trigger should be fired. Either on insert or update or delete or combination of any or all. More than one statement can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.

[OF columnname]:- This clause is used when you want to trigger an event only when a specific column is updated. This clause is mostly used with update triggers.

[ON table_name]:- This clause identifies the name of the table or view to which the trigger is related.

[REFERENCING OLD AS old NEW AS new]:- This clause is used to reference the old and new values of the data being changed. By default, you reference the values as old.column_name or new.column_name. The reference names can also be changed from old or new to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

[FOR EACH ROW]:- This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire SQL statement is executed(i.e.statement level Trigger).

WHEN (condition):- The trigger is fired only for rows that satisfy the condition specified. This clause is valid only for row level triggers.

Example 1

We are creating trigger that display message if we insert negative value or update value to negative value in bal column of account table.

```
CREATE OR REPLACE TRIGGER balnegative
        BEFORE insert OR update
On account
FOR EACH ROW
BEGIN
        IF :NEW.bal<0   THEN
        dbms.output.put.line ('balance is negative')
        END IF;
END;
```

OUTPUT:- Trigger is created.

- Now when you perform insert operation on account table.

```
SQL:>  Insert into account (bal) values (-2000);          OR
       Update account set bal=-5000 where bal=1000;
```

- It display following message before executing insert or update statement.

Output:- Balance is negative.

- We get message that balance is negative it indicate that trigger has executed before the insertion or update operation.

Example 2

```
CREATE TRIGGER trig1
   AFTER insert ON T4
                REFERENCING NEW AS newRow
 FOR EACH ROW
   WHEN (newRow.a <= 10)
BEGIN
                INSERT INTO T5 VALUES (newRow.a, newRow.b);
END;
```

OUTPUT:- Trigger is created.

Explanation:- We have following two tables

- ✓ CREATE TABLE T4 (a INTEGER, b CHAR(10));
- ✓ CREATE TABLE T5 (c INTEGER, d CHAR(10));

- We have created a trigger that may insert a tuple into T5 when a tuple is inserted into T4. It will check that the record insert in T4 is 10^{th} or less than 10^{th} record if so then the same record is inserted into table T5.

Example 3

```
CREATE TRIGGER Person_Check_Age
        AFTER insert OR update OF age ON Person
        FOR EACH ROW
BEGIN
        IF (:new.age < 0) THEN
        dbms.output.put.line('no negative age allowed');
        END IF;
END;
```

OUTPUT:- Trigger is created.

Explanation:-In above trigger if we are inserting negative value in Age column then it will give error. If we insert negative value in other column then it will not give any error.

Write short note on cursors in PL/SQL. OR Write short note on cursors and its types.

- Cursors are database objects used to traverse the results of a select SQL query.
- It is a temporary work area created in the system memory when a select SQL statement is executed.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- It points to a certain location within a recordset and allow the operator to move forward (and sometimes backward, depending upon the cursor type).
- We can process only one record at a time.
- The set of rows the cursor holds is called the *active* set (active data set).
- Cursors are often criticized for their high overhead.
- There are two types of cursors in PL/SQL:

1. **Implicit cursors:**
   - These are created by default by ORACLE itself when DML statements like, insert, update, and delete statements are executed.
   - They are also created when a SELECT statement that returns just one row is executed.
   - We cannot use implicit cursors for user defined work.
2. **Explicit cursors:**
   - Explicit cursors are user defined cursors written by the developer.
   - They can be created when a SELECT statement that returns more than one row is executed.
   - Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row.
   - When you fetch a row, the current row position moves to next row.

Attributes of Cursor:-

Attributes	Return Value	Example
%FOUND	It will return TRUE, if the DML statements like INSERT, DELETE, UPDATE and SELECT will affect at least one row else return FALSE	SQL%FOUND
%NOTFOUND	It will return FALSE, if the DML statements like INSERT, DELETE, UPDATE and SELECT will affect at least one row else return TRUE	SQL%NOTFOUND
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT
%ISOPEN	It will return true if cursor is open else return false.	SAL%OPEN

Steps to manage explicit cursor:-

1) Declare a cursor:-

A cursor is defined in the declaration section of PL/SQL block.

Syntax:-

```
CURSOR cursorname IS SELECT ……
```

2) Open a cursor:-

Once cursor is declared we can open it.

When cursor is opened following operations are performed.

a. Memory is allocated to store the data.
b. Execute SELECT statement associated with cursor
c. Create active data set by retrieving data from table
d. Set the cursor row pointer to point to first record in active data set.

Syntax:-

```
OPEN cursorname;
```

3) Fetching data:-

We cannot process selected row directly. We have to fetch column values of a row into memory variables. This is done by FETCH statement.

Syntax:-

FETCH cursorname INTO variable1, variable2…….

4) Processing data:-

This step involves actual processing of current row.

5) Closing cursor:-

A cursor should be closed after the processing of data completes. Once you close the cursor it will release memory allocated for that cursor.

Syntax:-

CLOSE cursorname;

Example:-

```
DECLARE CURSOR emp_cur IS SELECT emp_rec FROM emp_tbl WHERE salary > 10000;
BEGIN
OPEN emp_cur;
FETCH emp_cur INTO emp_rec;
dbms_output.put_line (emp_rec.first_name || ' ' || emp_rec.last_name);
CLOSE emp_cur;
END;
```

Explanation:-

- In the above example, first we are delaring a cursor whose name is emp_cur with select query.
- In select query we have used where condition such that salary<10000. So active data set contain only such records whose salalry less than 10000.
- Then, we are opening the cursor in the execution section.
- Then we are fetching the records from cursor to the variable named emp_rec.
- Then we are displaying the first_name and last_name of records fetched in variable.
- At last as soon as our work completed e are closing the cursor.