GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

HANDOUT

on

DATABASE MANAGEMENT SYSTEMS

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behaviour & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

PEO1 : Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2 : Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3 : Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON DATABASE MANAGEMENT SYSTEM

II B.Tech – II Semester Year: 2019-20

Branch: **CSE** Credits: **3**

==========

- **Brief History and Scope Of The Subject**

  Database is an organized collection of data. It is the collection of schemas, tables, queries, reports, views and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies. A **database management system** (**DBMS**) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases

- **Pre-Requisites**

  Students need to be aware of different storage mechanisms used for data storage

- **Course Objectives:**

  a. To familiarize the concepts of database systems and different issues involved in the database design.

  b. To introduce how to write SQL for storage,retrieval and manipulation of data in a relational database.

- **Course Outcomes:**

  Upon successful completion of the course, the students will be able to

  CO1: recognize the importance of database system over file processing system.

  CO2: analyze an information storage problem and derive an information model in the form of an entity relationship diagram.

  CO3: write simple and complex queries using structured query Language(SQL) for storage, retrieval and manipulation of data in a relational database.

CO4: employ principles of normalisation for designing a good relational database schema.

CO5: describe the issues and techniques relating to concurrency and database recovery in a multi-user database environment.

Program Outcomes:

Graduates of the Computer Science and Engineering Program will have Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and wit society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

- **Mapping of Course Outcomes with Program Outcomes:**

	1	2	3	4	5	6	7	8	9	10	11	12
CO1		3			2							
CO2	2	2	1	3								
CO3	3	3		2							1	
CO4	2		3		1							
CO5	2		3		1							

3- High **2- Medium** **1-Low**

- **Prescribed Text Books**
  1. Korth & Sudarshan, Database system concept, MH.
  2. Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, MH.
- **Reference Text Books**
  1. Elmasri Navrate, Fundamentals of Database Systems, Pearson Education.
  2. C.J.Date, Introduction to Database Systems, Pearson Education.

3. Peter Rob & C Coronel, Database Systems design, Implementation, and Management, 7th Edition.

- **URLs and Other E-Learning Resources**
  1. **http://www.w3schools.com/sql/**
  2. **http://www.mysqltutorial.org/**
  3. **http://www.java2s.com/Tutorial/Oracle/CatalogOracle.htm**
  4. **http://www.oracle.com/technetwork/tutorials/index.html**
- **Digital Learning Materials:**
  1. https://www.youtube.com/watch?v=rbwXdTsCk2c
  2. https://www.youtube.com/watch?v=EUzsy3W4I0g
- **Lecture Schedule / Lesson Plan**

TOPIC	No. of Periods	
	Theory	Tutorial
UNIT –1: INTRODUCTION TO DATABASE		
Advantages of using DBMS	1	2
Data Models, Schema and instances	2	
Levels of abstraction	1	
Entity- Relationship Model- Attributes and Keys	1	
Relationship Types,Weak Entity set, Strong Entity Set	1	
Enhanced E–R Modeling- specialization,generalization	1	
database design for banking enterprise	1	
reduction to relational schemas	3	
UNIT – 2: RELATIONAL MODEL & SQL		
Relational model concepts	1	2
constraints, keys, relational algebra	1	
SQL- DDL, DML	1	
Set operations, Aggregate Functions	1	
Null values, Nested queries	2	
Defining different constraints on a table	2	
Group by, having clauses	1	

UNIT – 3: DATABASE DESIGN		
Functional Dependencies: partial, full, transitive dependencies	2	3
Axioms	1	
attribute closure	2	
Lossless join, dependency preserving decomposition	2	
Normal forms-First, second third normal forms	2	
Boyce- Codd normal form	1	
UNIT – 4: TRANSACTION MANAGEMENT		
Transaction Management- Transaction concept	1	2
ACID properties, transaction state diagram	2	
Schedules: serial, concurrent, serializable	2	
serializability of schedules	3	
recoverability	2	
UNIT – 5: CONCURRENCY CONTROL		
Concurrency control	1	3
Concurrent execution of transactions, anomalies	2	
Lock-based protocols:2PL, strict 2PL, rigorous 2PL	1	
Timestamp-based protocols	2	
Thomas write rule, Deadlock handling: deadlock prevention	2	
Deadlock detection and recovery	1	
UNIT – 6: CRASH RECOVERY		
Failure classification	1	2
Different types of Recovery techniques-deferred update, immediate update	3	
Shadow paging, Check points	3	
Total No. of Periods:	**56**	**14**

- **Seminar Topics**
  - Serializability
  - Normalization
  - Transaction Management
  - Lock-based protocols
  - Crash recovery techniques

UNIT-1
DATABASE MANAGEMENT SYSTEMS

Objectives:

To provide the basic knowledge about the database management system

Syllabus:

Introduction to Database Purpose of Database Systems, Data models, Schema and instances, DBMS architecture, Entity- Relationship model- Attributes and Keys, Relationship types, Weak Entity set, Strong Entity Set, enhanced E–R Modelling- Specialization and generalization, Database design for Banking enterprise, reduction to relational schemas.

Learning Outcomes:

At the end of the unit, students will be able to

- differentiate database systems from file systems by enumerating the features provided by the database system.
- describe fundamental elements of a database management system.
- design entity relationship diagrams to represent simple database application scenarios.
- convert entity relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data.
- criticize a database design and improve the design by normalization.
- interpret the basic issues of transaction processing and concurrency control.

- **Introduction to Database Management Systems**
  - **Database -** Database is a large integrated collection of data
  - **Database Management System (DBMS) –** Database management system is a collection of interrelated data and a set of programs to

access those data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

- Databases are usually designed to manage large bodies of information. This involves
    - Definition of structures for information storage (data modeling).
    - Provision of mechanisms for the manipulation of information (File and systems structure, query processing).
    - Providing for the safety of information in the database (crash recovery and security).
    - Concurrency control if the system is shared by users.

- **Purpose of Database Systems**
    - To see why database management systems are necessary, let's look at a typical file processing system supported by a conventional operating system.
    - The application is a savings bank:
        - Savings account and customer records are kept in permanent system files.
        - Application programs are written to manipulate files to perform the following tasks:
            - Debit or credit an account.
            - Add a new account.
            - Find an account balance.
            - Generate monthly statements.
    - Development of the system proceeds as follows:

- New application programs must be written as the need arises.
- New permanent _les are created as required

- But over a long period of time files may be in different formats, and Application programs may be in different languages.
- So we can see there are problems with the straight file processing approach:
- **Data redundancy and inconsistency**
  - Same information may be duplicated in several places.
  - All copies may not be updated properly.
- **Difficulty in accessing data**
  - May have to write a new application program to satisfy an unusual request.
  - E.g. find all customers with the same postal code.
  - Could generate this data manually or need to write an application program for getting the desired information. Both these activities are very difficult.
- **Data isolation**
  - Data in different files.
  - Data in different formats.
  - Difficult to write new application programs to access the data
- **Integrity problems**
  - Data may be required to satisfy constraints.
  - E.g. no account balance below $25.00.

- Again, difficult to enforce or to change constraints with the file-processing approach.

- **Atomicity Problems**
  - A computer system, like any other mechanical or electrical device, is subject to failure.
  - In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.
  - E.g. Consider a program to transfer $500 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $500 was debited from account A but not credited to account B, resulting in an inconsistent database state.
  - Clearly, it is essential to database consistency that either both credit or debit occurs or that neither occurs. i.e. the funds transfer must be atomic – it must happen in its entirety or not at all.
  - It is difficult to ensure atomicity in a conventional file processing system.

- **Concurrent Access anomalies**
  - Want concurrency for faster response time.
  - Need protection for concurrent updates.
  - E.g. two customers withdrawing funds from the same account at the same time - account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 rather the correct value of $350.

- It is very difficult to guard against concurrent executions in file processing system because data may be accessed by many different application programs that have not been coordinated previously.

- **Security problems**
  - Every user of the system should be able to access only the data they are permitted to see.
  - E.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.
  - Difficult to enforce this with application programs.

These problems and others led to the development of database management systems.

- **Data Abstraction**
  - The major purpose of a database system is to provide users with an abstract view of the system. The system hides certain details of how data is stored and maintained.
  - Complexity should be hidden from database users.
  - There are three levels of abstraction:

Physical Level

- Deals with how the data are stored.
- E.g. index, B-tree, hashing.
- Lowest level of abstraction.
- Complex low-level structures described in detail.

Conceptual Level

- Next highest level of abstraction.
- Describes what data are stored.
- Describes the relationships among data.
- Database administrator level.

View Level

- Highest level.
- Describes part of the database for a particular group of users.
- Can be many different views of a database.
- E.g. tellers in a bank get a view of customer accounts, but not of payroll data.

Figure 1.1 illustrates three levels of abstraction.

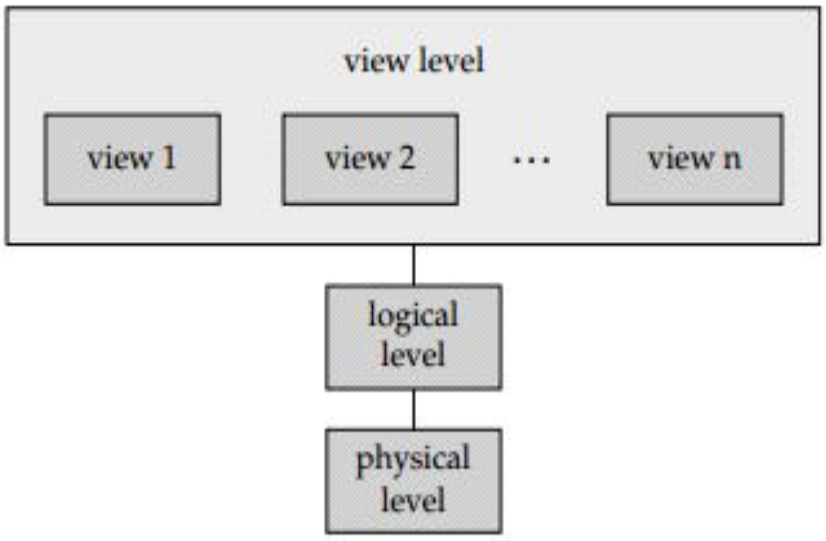


Figure 1.1 The three levels of data abstraction.

- **Data models**
  - Data models are a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. Some of the popular data models are:

Entity-Relationship model

- In this, data is represented as a collection of entities and relationships among the entities.
- An entity is a thing or object in the real world that is distinguishable from other objects.

- Each entity is described by set of attributes.
- E.g. ***Customer*** entity is described by the attributes *customer-id*, *customer-name*, *customer-street,* and *customer-city*
- A relationship is an association among several entities
- E.g. A ***borrower*** relationship associates a Customer entity with Account entity
- The set of all entities of the same type is called as the entity set.
- Similarly, the set of all relationships of the same type is called as the relationship set.
- The overall logical structure of a database can be expressed graphically by an E-R diagram where:
  - **rectangles:** represent entity sets.
  - **ellipses:** represent attributes.
  - **diamonds:** represent relationships among entity sets.
  - **lines:** link attributes to entity sets and entity sets to relationships.

Figure 1.2 illustrates a sample ER diagram.

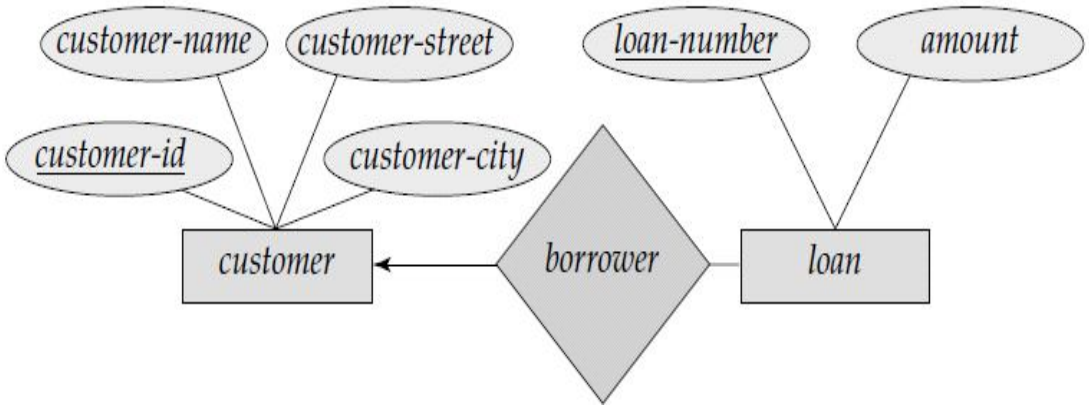


Fig. 1.2: a sample ER diagram

Relational Model

- In this, data is represented as a collection of tables also known as relations. E.g. customer table, account table
- Each table has a number of columns with unique names. E.g. customer table has *customer-id*, *customer-name*, *customer-street*, and *customer-city columns*.
- Similarly, loan table has *loan-number* and *amount* columns.
- Figure 1.3 shows a sample relational model.

customer_id	*customer_name*	*customer_street*	*customer_city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

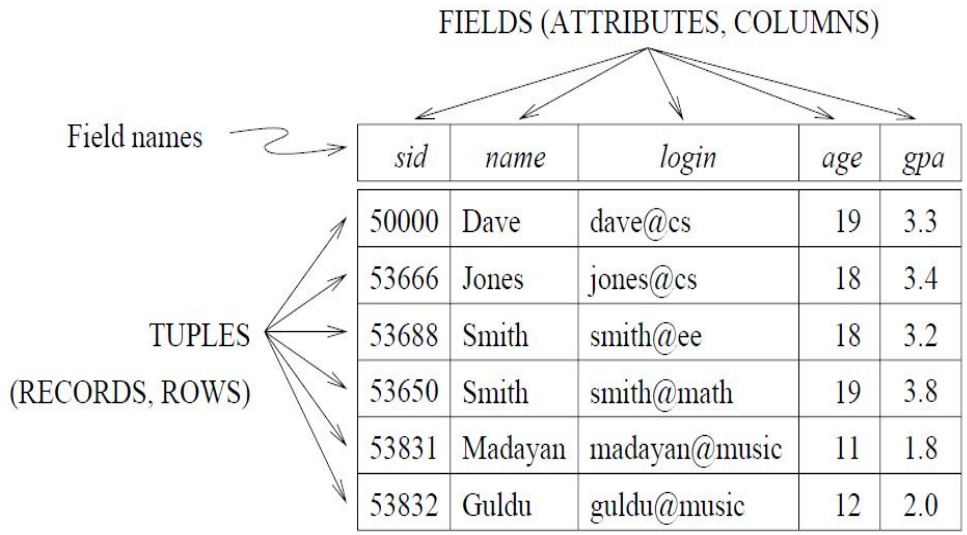
(b) The *account* table

customer_id	*account_number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Figure 1.3: A sample Relational Model.

- The degree, also called arity, of a relation is the number of fields in it.
- The cardinality of a relation is the number of tuples in it.
- For example, for the relation shown in figure 1.4, the degree of the relation is 5, and cardinality is 6.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Fig. 1.4: student relation

Network Model

- In this, data are represented by collections of records.
- Relationships among data are represented by links.
- Collection of records are organised as an arbitrary graph.
- Figure 1.5 shows a sample network model.

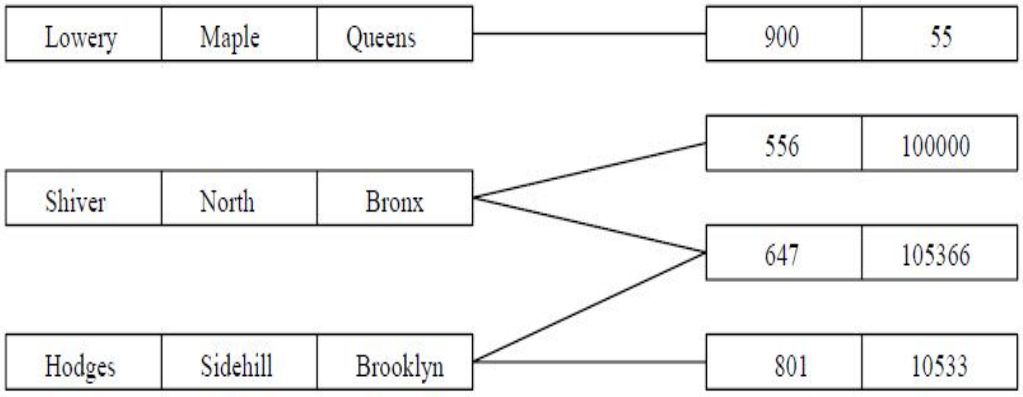


Fig. 1.5: A sample Network Model

Hierarchical Model

- It is similar to the network model.
- Organizes the collection of the records as trees, rather than arbitrary graphs.
- Figure 1.6 shows a sample hierarchical database.

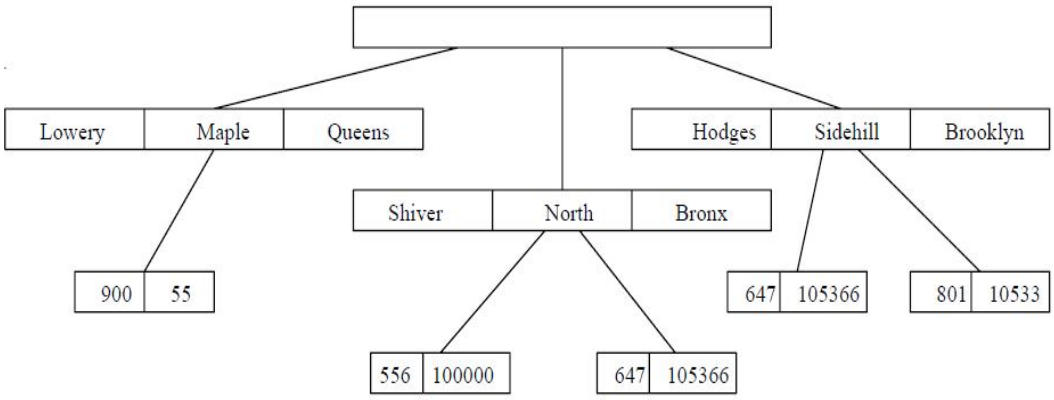


Fig. 1.6: A sample Hierarchical Model.

Object Relational Model

- Combines the features of the object-oriented data model and relational data model.

- **Instances and Schemas**
  - Databases change over time.
  - The information in a database at a particular point in time is called an instance of the database.
  - The overall design of the database is called the database schema. In other words, a database schema consists of collection of relational schemas.
  - The overall structure of the relation (or table) is known as relation schema. E.g. student relation schema is shown as

    student (sid, name, login, age, gpa)

- The information in a relation (or table) at a particular point in time is called an instance of the relation. E.g. figure 1.7 shows instance of the *student relation.*

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Fig. 1.7: Instance of *student relation*

Fig. 1.8: Database system structure

- **Entity-Relationship Model**
  - In this, data is represented as a collection of entities and relationships among the entities.
  - An entity is a thing or object in the real world that is distinguishable from other objects.
  - Each entity is described by set of attributes.
  - E.g. ***Customer*** entity is described by the attributes *customer-id*, *customer-name*, *customer-street,* and *customer-city*
  - A relationship is an association among several entities
  - E.g. A ***borrower*** relationship associates a Customer entity with Account entity
  - The set of all entities of the same type is called as the entity set.

- Similarly, the set of all relationships of the same type is called as the relationship set.
- The overall logical structure of a database can be expressed graphically by an E-R diagram where:
  - **rectangles:** represent entity sets.
  - **ellipses:** represent attributes.
  - **diamonds:** represent relationships among entity sets.
  - **lines:** link attributes to entity sets and entity sets to relationships.
  - **Double ellipses:** represent multivalued attributes.
  - **Dashed ellipses:** denote derived attributes
  - **Double lines:** indicate total participation of an entity in a relationship set (described later).
  - **Double rectangles:** represent weak entity sets (described later)

Figure 1.9 shows a sample ER diagram.

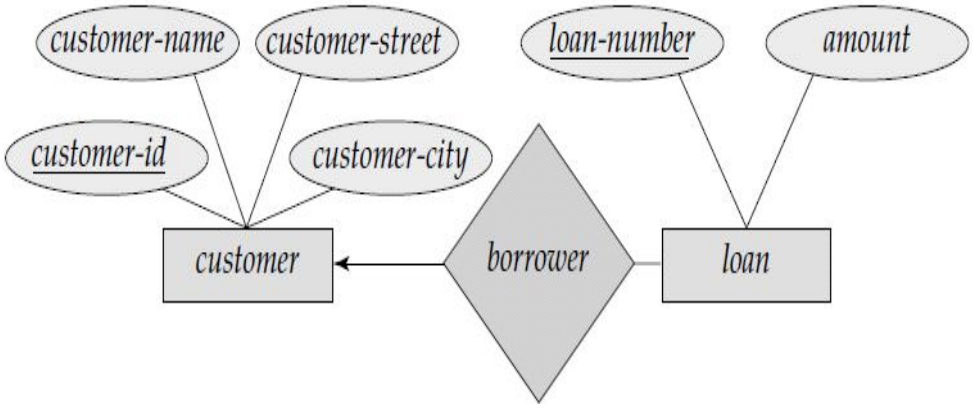


Fig. 1.9: A sample E-R diagram

7.1 Attributes

- Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

- Attributes are categorized into following types:
- Simple attribute – is an attribute that cannot be divided into sub parts.
  E.g. account-number
- Composite attribute – is an attribute that can be divided into sub parts.
  E.g. customer-name (It can be divided into sub parts as first_ name, middle _name and last_ name).

The following figure (Figure 1.10) shows composite attributes.

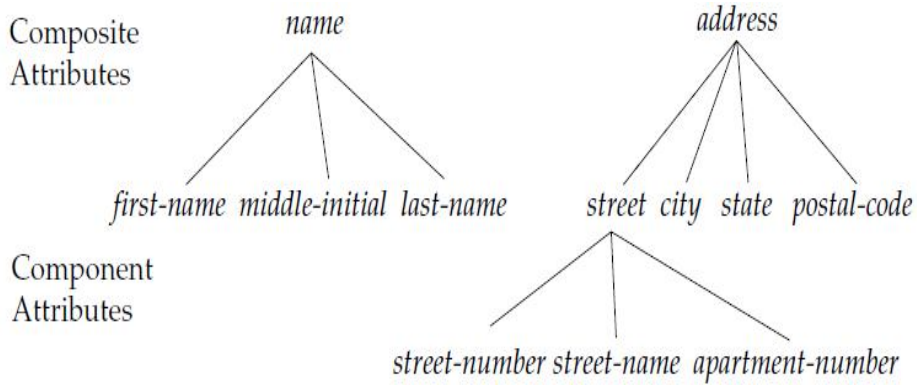


Fig. 1.10: Composite attributes

- **Single valued attribute:** an attribute that takes only one value.
  E.g. rollnumber
- **Multi valued attribute:** an attribute that takes more than one value.
  E.g. phone-number
- **Derived attribute:** attribute can be derived from the values of other related attributes.
  E.g. age is a derived attribute as its value can be derived from date_of_birth attribute.

The following figure (Figure 1.11) shows an ER diagram containing composite, multivalued and derived attributes.

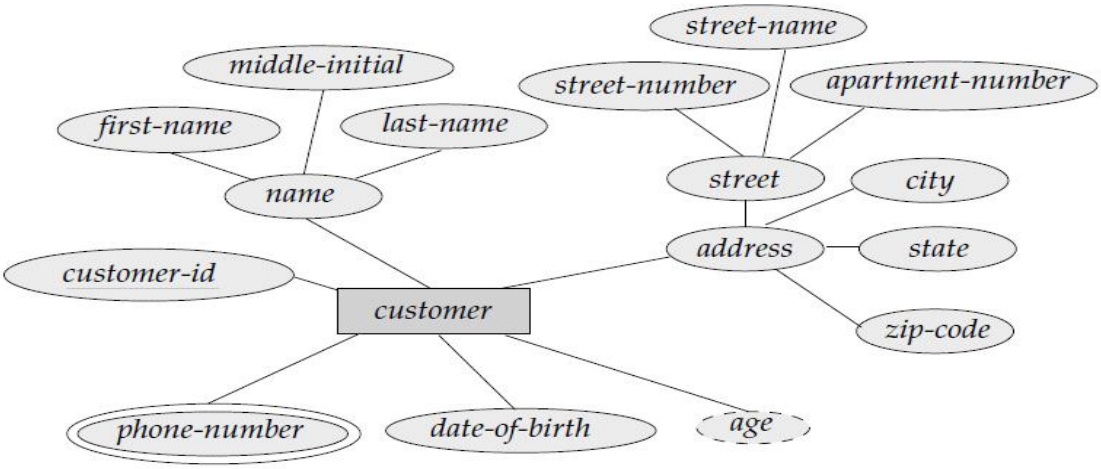


Fig. 1.11: E-R Diagram that shows composite, multivalued and derived attributes.

7.2 Keys

- Key is a set of attributes that uniquely identify an entity set from set of entities.
- E.g. suppose student entity consists of the attributes (*Rollno, Name, DOB, Address*) then the attribute *Rollno* identifies uniquely a student from set of students. Hence, *Rollno* is the key for student entity.

- **Relationship Types**
  - Relationship is an association among two or more entities.
  - There are three kinds of relationships
- **Unary Relationship** – is a relationship that associates an entity with the same entity. E.g. Figure 1.12 shows Unary Relationship(works-for)

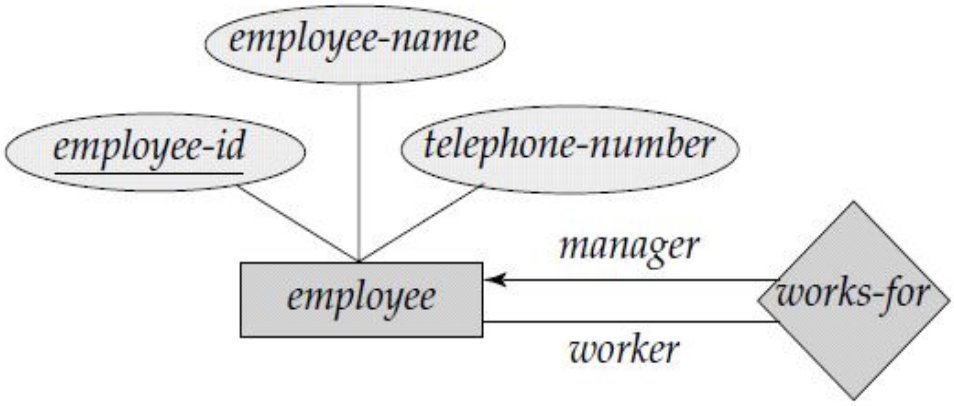


Fig. 1.12: An example Unary Relationship.

- **Binary Relationship** - is a relationship between any two entities.
  E.g. Figure 1.13 shows a binary relationship (depositor) between the entities *customer* and *account*.

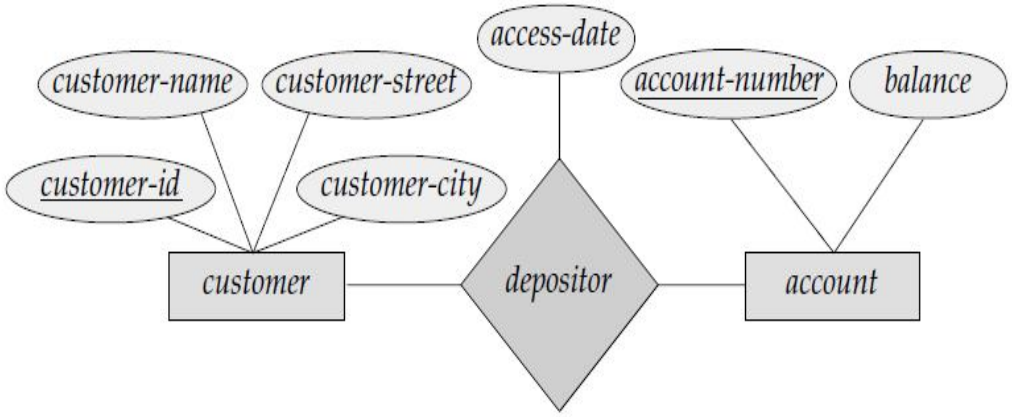


Fig. 1.13: An example Binary Relationship.

- **Mapping Cardinalities** - Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets.
- For a binary relationship R between entity sets A and B, the mapping cardinalities must be one of the following:
  - **One to one.** An entity in A is associated with at most one entity in B. and an entity in B is associated with at most one entity in A.
  - **One to many.** An entity in A is associated with any number of entities in B. An entity in B, however, can be associated with at most one entity in A.

  Figure 1.14 shows one-to-one and one-to-many mapping cardinalities.

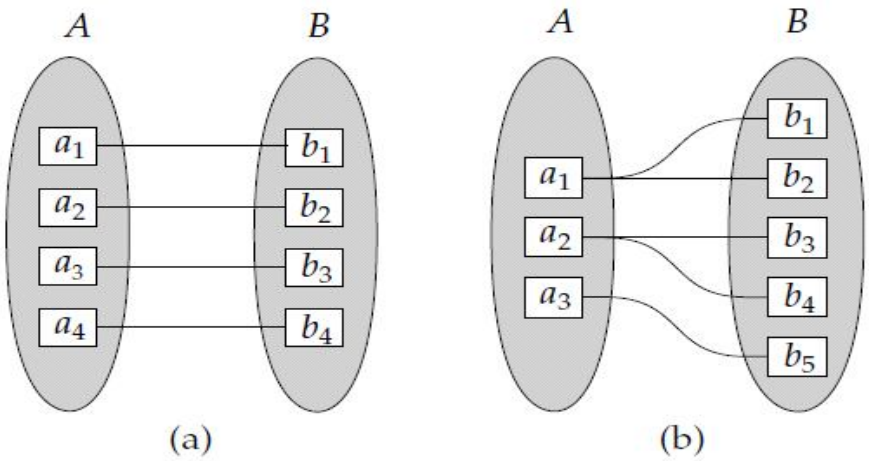


Fig. 1.14: Mapping cardinalities (a) one to one (b) one to many

- **Many to one.** An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.

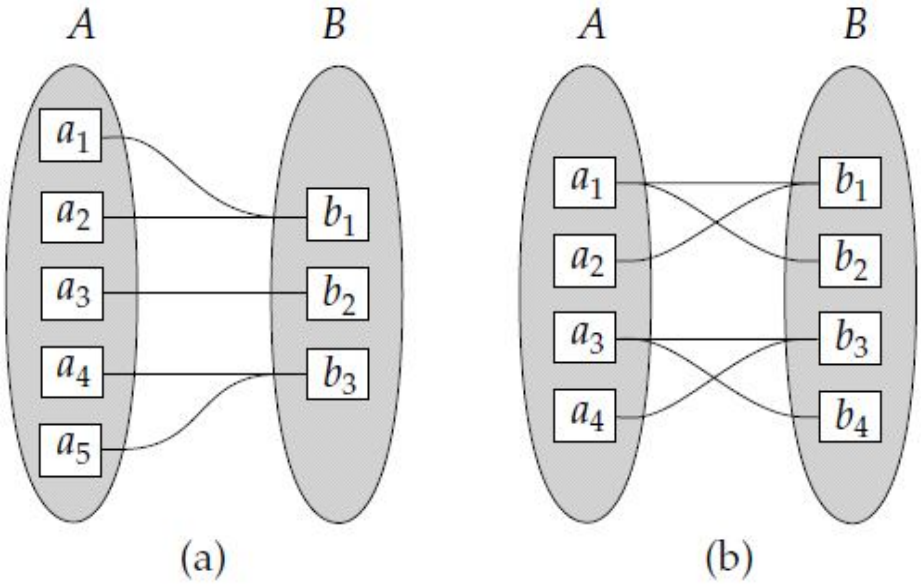


Fig. 1.15: Mapping cardinalities (a) many to one (b) many to many

- **Many to many**. An entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A.

Figure 1.15 shows many-to-one and many-to-many mapping cardinalities.

- **Ternary Relationship** A ternary relationship is one where three entities participate in the relationship. Figure 1.16 shows a ternary relationship.

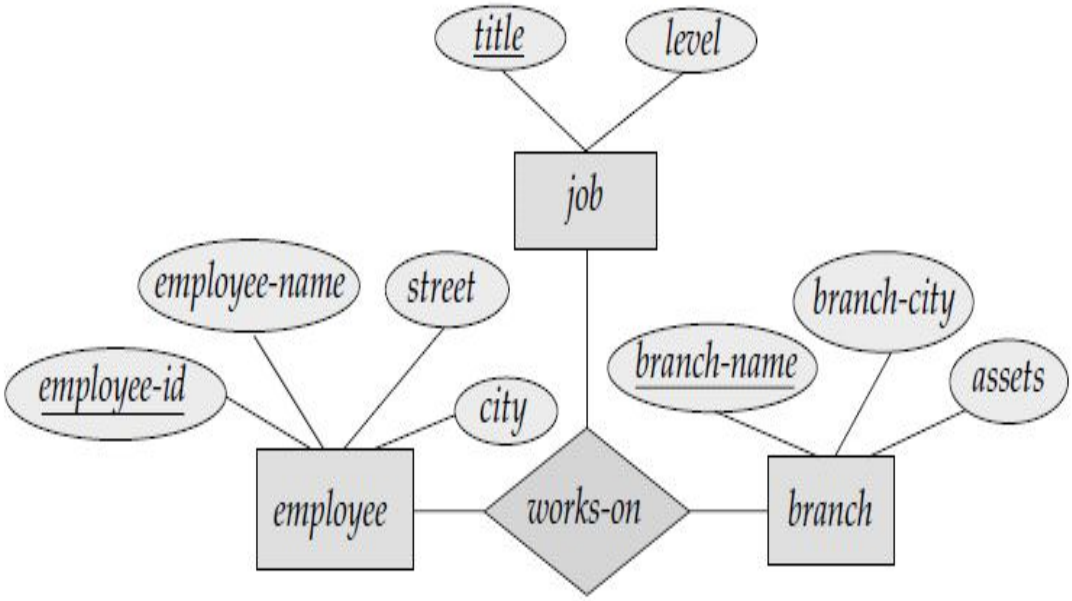


Fig. 1.16: An example ternary relationship.

7.4 Participation constraints.

- Participation constraints describe how the entities are participating in the relationship.
- There are two types of participation constraints.
  - Total Participation – Every entity in the entity set participates in the relationship.
  - Partial Participation – Only some of the entities in the entity set participates in the relationship.

The following diagram (Figure 1.17) shows participation constraints.

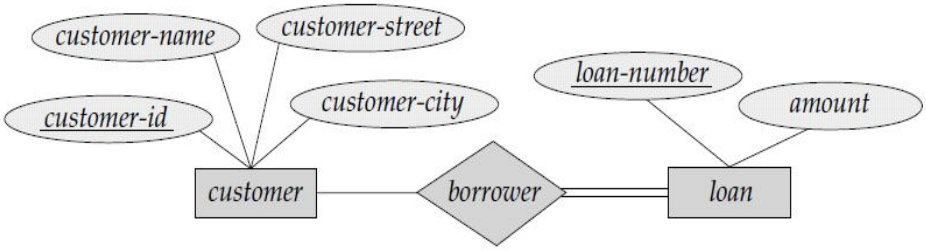


Fig. 1.17: ER diagram showing participation constraints.

- In this diagram, only some of the customers borrow loan hence, customer participation in borrower relationship is partial. Where as for every loan there is a borrower hence, loan participation in borrower relationship is total.

- **Weak entity set –** Entity set without primary key is termed as a weak entity set.
  - A weak entity set can be identified uniquely only by considering some of its attributes in conjunction with the primary key of another entity, called the owner set.
  - Owner entity set and weak entity set must participate in a one-to-many relationship.
  - This relationship set is called the identifying relationship set of the weak entity set. The weak entity set must have total participation in the identifying relationship set.

- **Strong entity set -** An entity set that has a primary key is called as a strong entity set.

  E.g. Figure 1.18 shows weak and strong entity sets. Here, ***payment*** is a weak entity set and ***loan*** is a strong entity set. Primary key for ***loan*** is **loan-number** and primary key for ***payment*** is **{loan-number, payment-number}.**

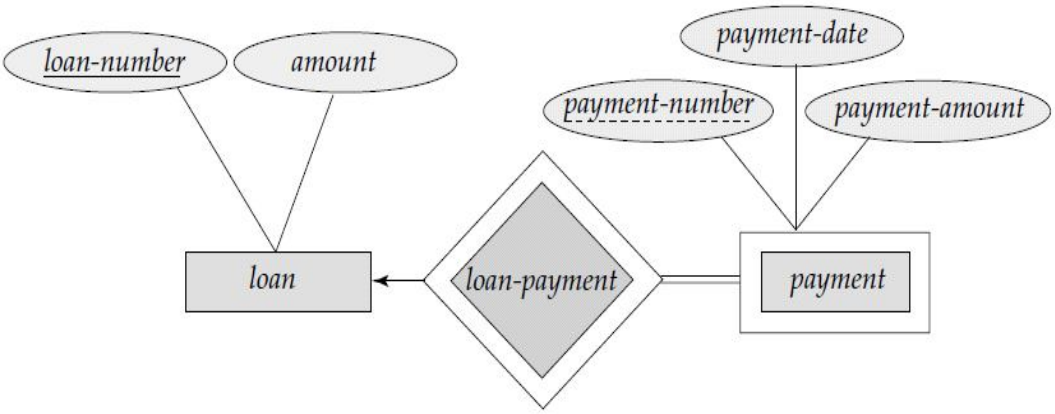


Figure1.18 Weak entity set.

- **Enhanced ER modeling**
  - Enhanced ER modelling includes specialization and generalization.
  - **Specialization** – The process of identifying subsets of an entity set that share some distinguishing characteristics is known as specialization.
  - It is also called as top-down design process.
  - Here, after identifying a general entity, its sub entities are identified that share the common characteristics of the general entity.
  - **Generalization** – The process of identifying some common characteristics of a collection of entity sets and creating a new entity set that contains entities possessing these common characteristics is known as generalization.
  - It is also called as bottom-up design process.
  - In ER diagram, generalization/specialization is denoted by ISA relationship.
  - We can inherit the properties of super class (general entity) into sub class (specialized entities) by ISA relationship.

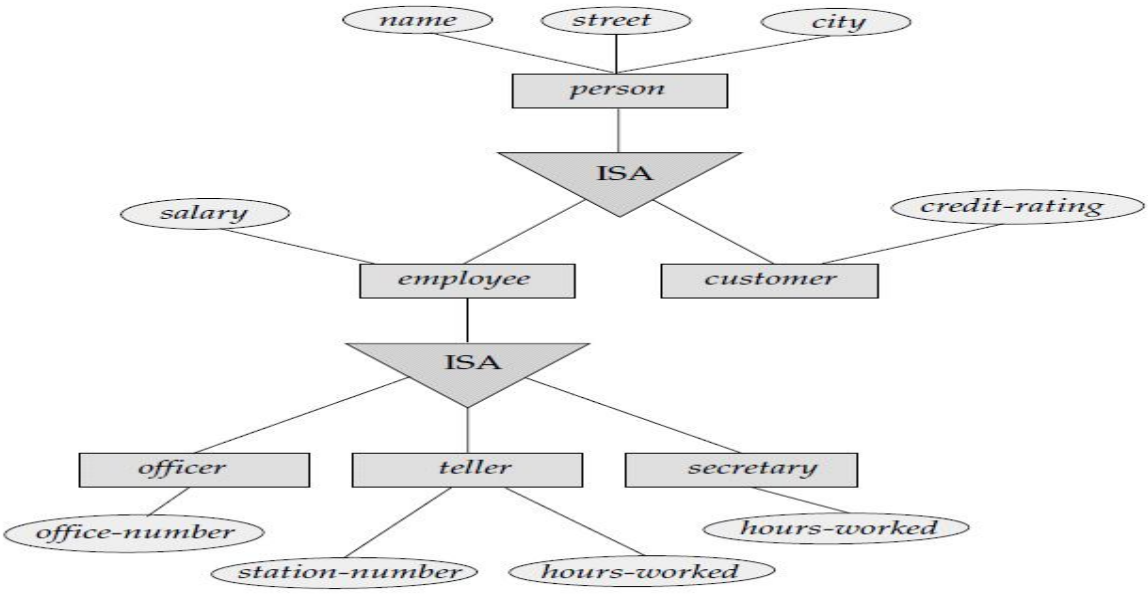


Fig. 1.19: An example of Specialization and Generalization.

- We can specify two kinds of constraints with respect to ISA hierarchies, namely, *overlap* and *covering* constraints. Overlap constraints determine whether two subclasses are allowed to contain the same entity. Covering constraints determine whether the entities in the subclasses collectively include all entities in the superclass.
  E.g. Figure 1.19 shows specialization and generalization.

Aggregation

- The E-R model cannot express relationships among relationships.
- Aggregation is used to express relationships among relationships
- As an example consider the following figure. Here, we treat the relationship set *work* and the entity sets *employee* and *project* as a higher-level entity set called ***work*** and this aggregate component can be related with ***machinery*** entity using ***uses*** relationship. It means that employee working on a project uses machinery.

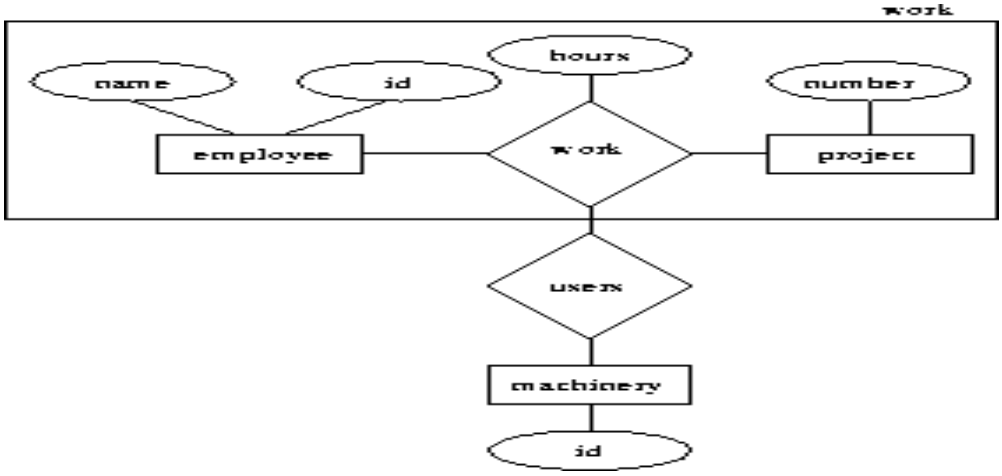


Fig. 1.20**:** E-R diagram with aggregation

- 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*.

- **Database Design For Banking Enterprise**
  - To design the database for Banking Enterprise, first the requirements of Banking Enterprise are gathered and specified. From this

requirement specification, the conceptual design of database is created. Here are the major characteristics of the banking enterprise;

- The bank is organized into branches. Each branch is located in a particular city and is identified by a unique name. The bank monitors the assets of each branch.
- Bank customers are identified by their *customer-id* values. The bank stores each customer's name, and the street and city where the customer lives. Customers may have accounts and can take out loans. A customer may be associated with a particular banker, who may act as a loan officer or personal banker for that customer.
- Bank employees are identified by their *employee-id* values. The bank administration stores the name and telephone number of each employee, the names of the employee's dependents, and the *employee-id* number of the employee's manager. The bank also keeps track of the employee's start date and, thus, length of employment
- The bank offers two types of accounts—savings and checking accounts. Accounts can be held by more than one customer, and a customer can have more than one account. Each account is assigned a unique account number. The bank maintains a record of each account's balance, and the most recent date on which the account was accessed by each customer holding the account. In addition, each savings account has an interest rate, and overdrafts are recorded for each checking account.
- A loan originates at a particular branch and can be held by one or more customers. A loan is identified by a unique loan number. For each loan, the bank keeps track of the loan amount and the loan payments. Although a loanpayment number does not uniquely identify a particular payment among those for all the bank's loans, a payment number does identify a

particular payment for a specific loan. The date and amount are recorded for each payment

- the bank would keep track of deposits and withdrawals from savings and checking accounts.

Keeping in mind the above requirements, the design of the database proceeds as follows;

Step1: Identify the entity sets and their attributes

The entity sets that could be identified for Banking Enterprise are:

- *branch* entity set, with attributes *branch-name*, *branch-city*, and *assets*
- *customer* entity set, with attributes *customer-id*, *customer-name*, *customer-street*, and *customer-city*
- *employee* entity set, with attributes *employee-id*, *employee-name*, *telephone-number*, *salary*, and *manager.* Additional descriptive features are the multi-valued attribute *dependent-name,* the base attribute *start-date,* and the derived attribute *employment-length*
- Two *account* entity sets—*savings-account* and *checking-account*—with the common attributes of *account-number* and *balance*; in addition, *savings-account* has the attribute *interest-rate* and *checking-account* has the attribute *overdraft-amount*
- The *loan* entity set, with the attributes *loan-number*, *amount*, and *originating-branch*
- The weak entity set *loan-payment*, with attributes *payment-number*, *payment-date*, and *payment-amount*

Step2: Identify the relationships and their cardinalities

- *borrower*, a many-to-many relationship set between *customer* and *loan*
- *loan-branch* between *loan* and *branch*, a many-to-one relationship set that indicates in which branch a loan originated
- *loan-payment*, a one-to-many relationship from *loan* to *payment*, which documents that a payment is made on a loan

- *depositor*, with relationship attribute *access-date*, a many-to-many relationship set between *customer* and *account*, indicating that a customer owns an account
- *cust-banker*, with relationship attribute *type*, a many-to-one relationship set expressing that a customer can be advised by a bank employee, and that a bank employee can advise one or more customers
- *works-for*, a relationship set between *employee* entities with role indicators *manager* and *worker*; the mapping cardinalities express that an employee works for only one manager and that a manager supervises one or more employees

Step3: Draw the E-R diagram with the identified entity sets and their relationships

Figure 1.21shows the ER diagram for banking enterprise.

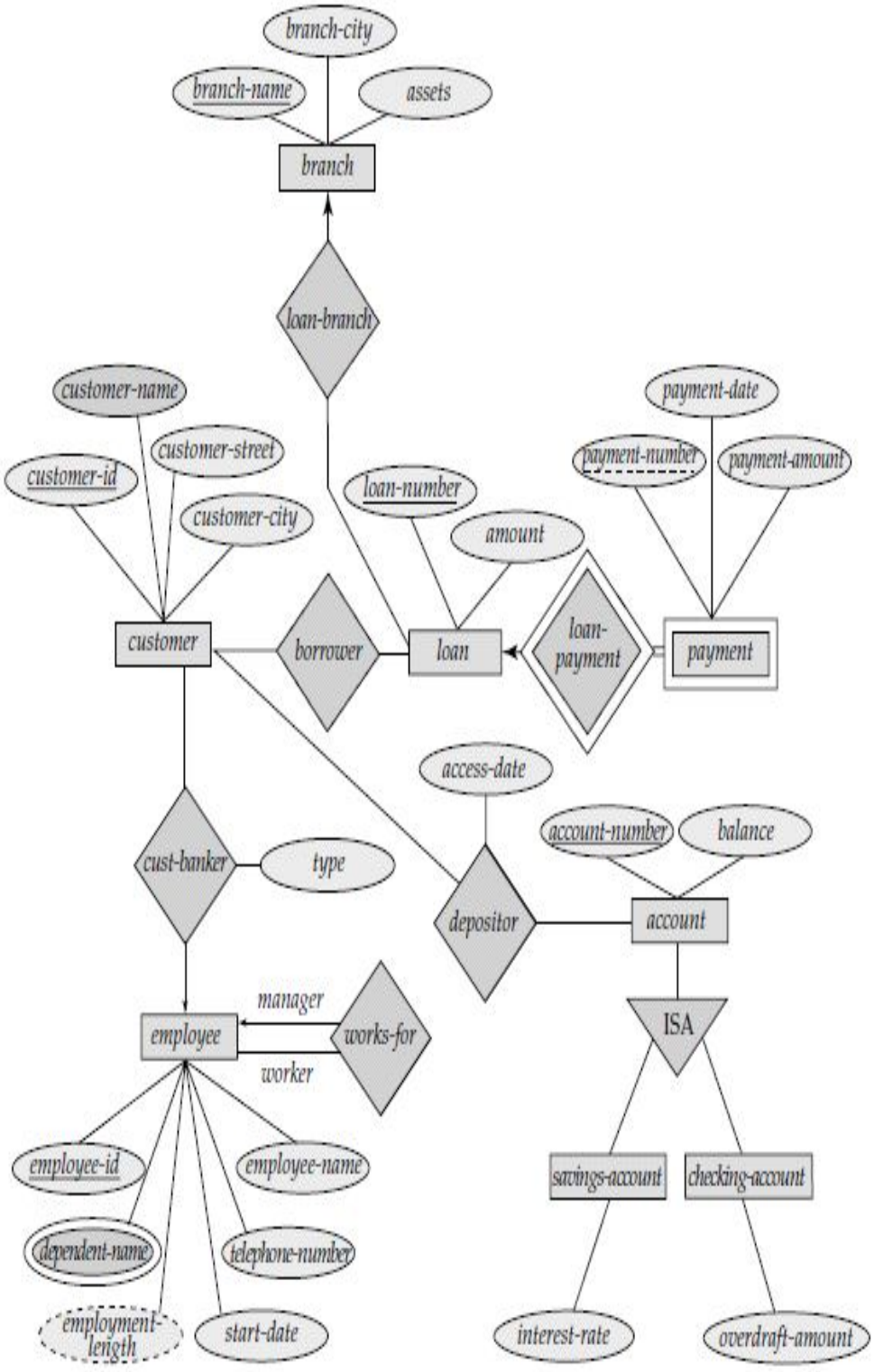


Fig. 1.21: ER diagram for banking enterprise

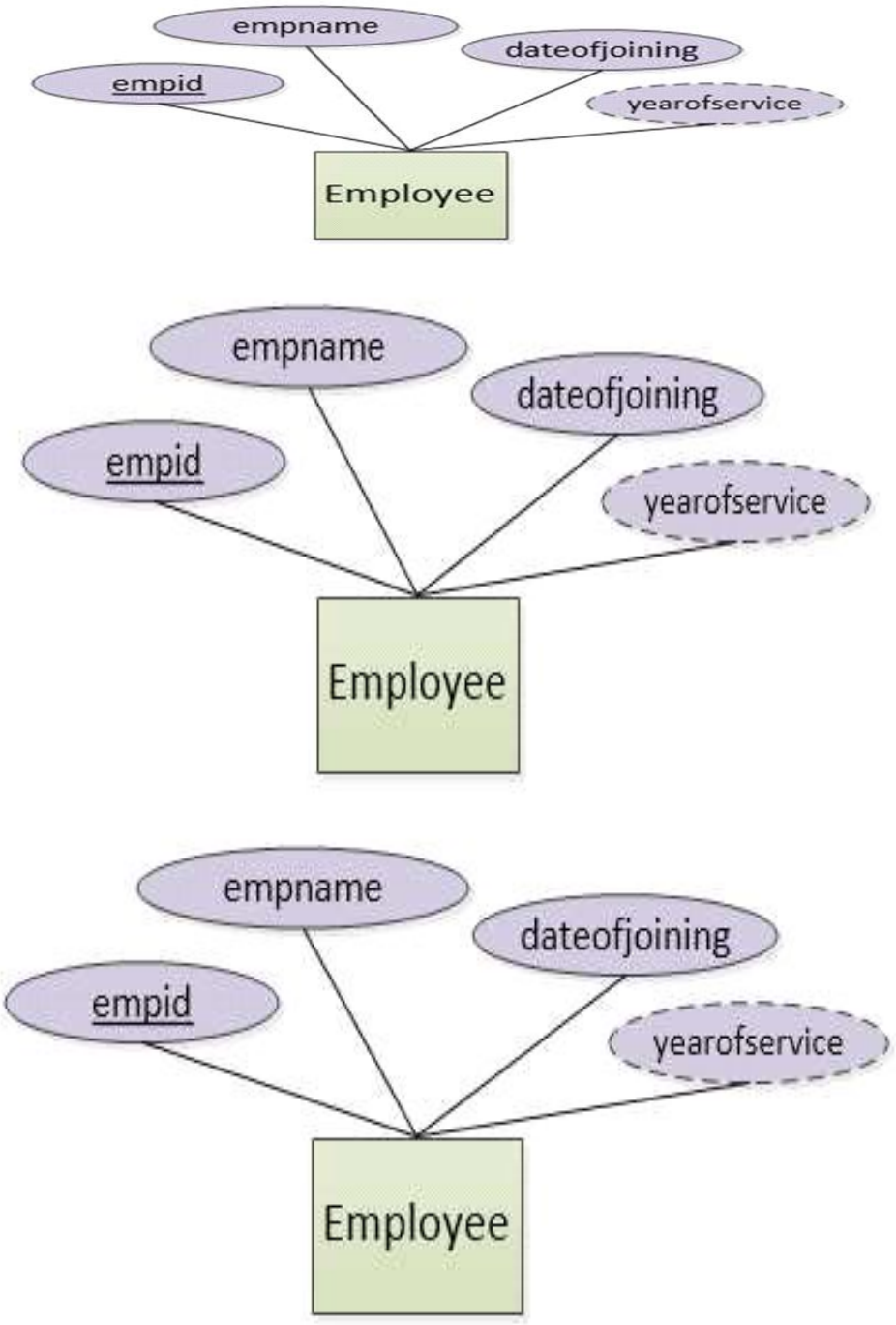
- **Reduction to Relational Schemas**

  Conversion Guidelines

  - Each entity in ER diagram becomes a table in relational schema.
  - Each single-
    valued attribute in ER diagram becomes a column of the table.
  - Derived attributes of entities are ignored.
  - Composite attributes of an entity are represented by its equivalent par ts.
  - Multi-valued attributes are kept in a separate table.
  - The key attribute of an entity is chosen as the primary key of the tabl e.
  - Converting relationships is based on degree and cardinality of relation ship.

- **Conversion of strong entity set**

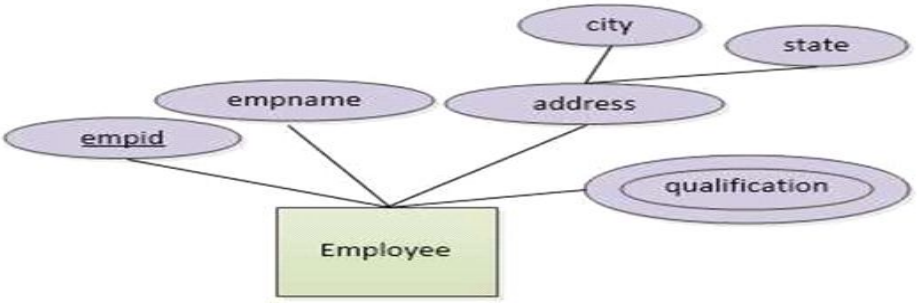
  Example 1:



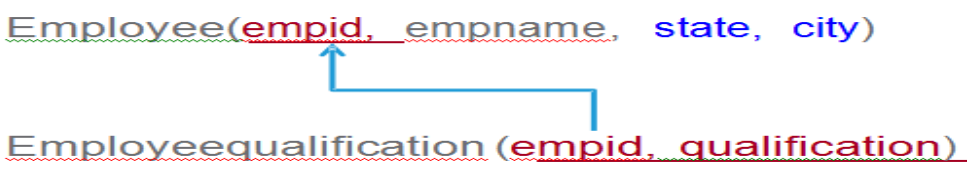
Relational Schema:

Employee(empid, empname, dateofjoining)

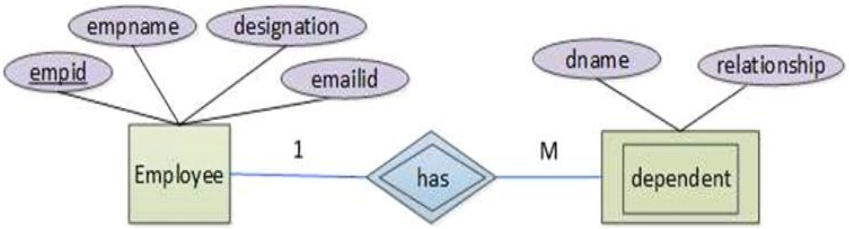
Example 2: Conversion of Strong Entity Set when it has multi-valued attribute



Relational Schema:



Example 3: Conversion of Weak entity set



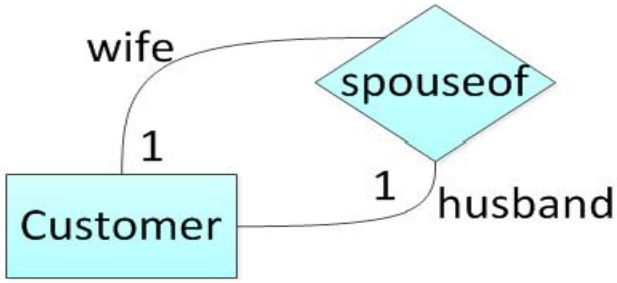
Relational Schema:

employee(empid,empname, designation, emailid)

dependent (empid, dname, relationship)

- **Conversion of Unary Relationship (1:1)**

Example 1:

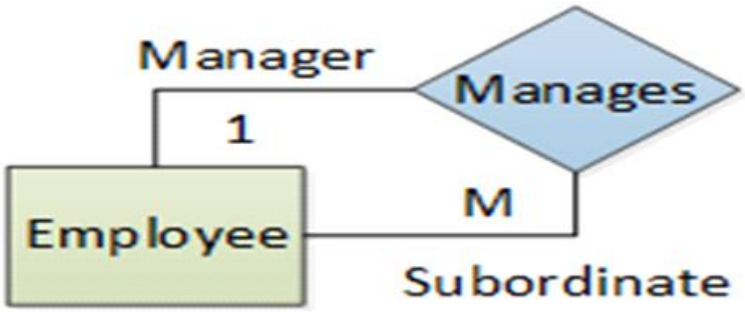


Relational Schema:

customer(customerid, customername,…spouse)

Example 2: Unary Relationship (1:M)

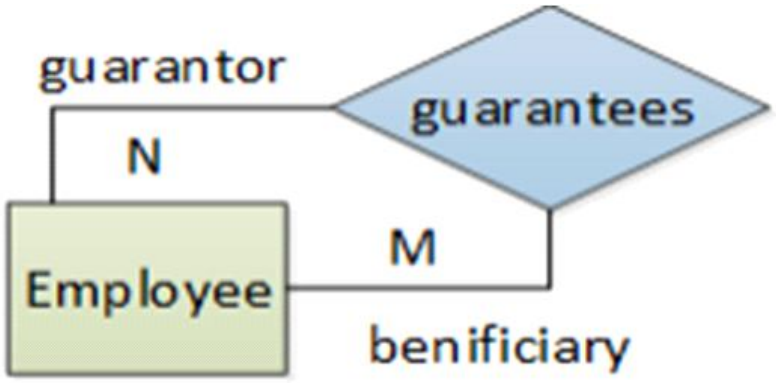
The primary key of the table will itself become foreign key of t he same table



Relational Schema:

Employee(empid, empname, dateofjoining, designation, ……, managerid)

Example 3: Unary Relationship (N:M)



Relational Schema:

employee(empid, empname, designation,....)

guaranty(guarantorid, beneficiaryid)

- **Conversion of Binary Relationship (1:1)**

  Example 1:

  The key attribute of any of the participating entities in a relationship can become a foreign key in the other participating entity.

Relational Schema:

employee(empid, empname, designation, ..., salary)

retailoutlet (retailoutletid, retailoutletlocation, retailouletmanagerid)

OR

employee(empid, empname, designation,.....,salary, retailoutletid)

retailoutlet (retailoutletid, retailoutletlocation)

Example 2: Binary Relationship (1: M)

The key attribute of entity on the "1" side of the relationship becomes a foreign key of entity towards the "M" side.

Relational Schema:

supplier (supplierid, suppliername, suppliercontactno, supplieremailid)

quotation (quotationid, itemcode, quotedprice, supplierid)

Example 3: Binary Relationship (N: M)

Relational Schema:

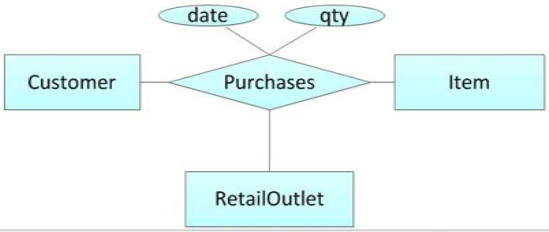
customer (customerid,customertype,customername,emailid,contactno,address)

retailoutlet (retailoutletid, retailoutletlocation)

purchasesfrom (customerid, retailoutletid)

- **Conversion of Ternary relationship**

Example 1:



Relational Schema:

customer (customerid,customertype,customername,emailid,contactno,address)

retailoutlet (retailoutletid, retailoutletlocation)

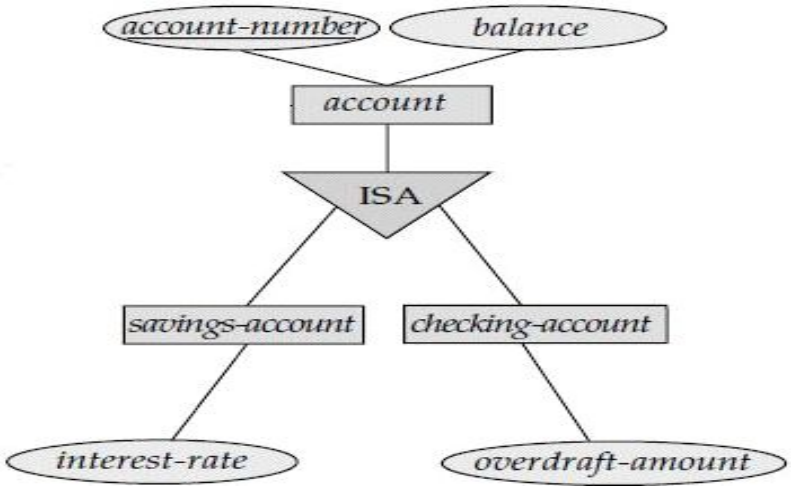
item (itemid, description, reorderlevel, itemclass)

purchases(customerid, retailoutletid, itemcode, date, qty)

- **Conversion of Generalization**

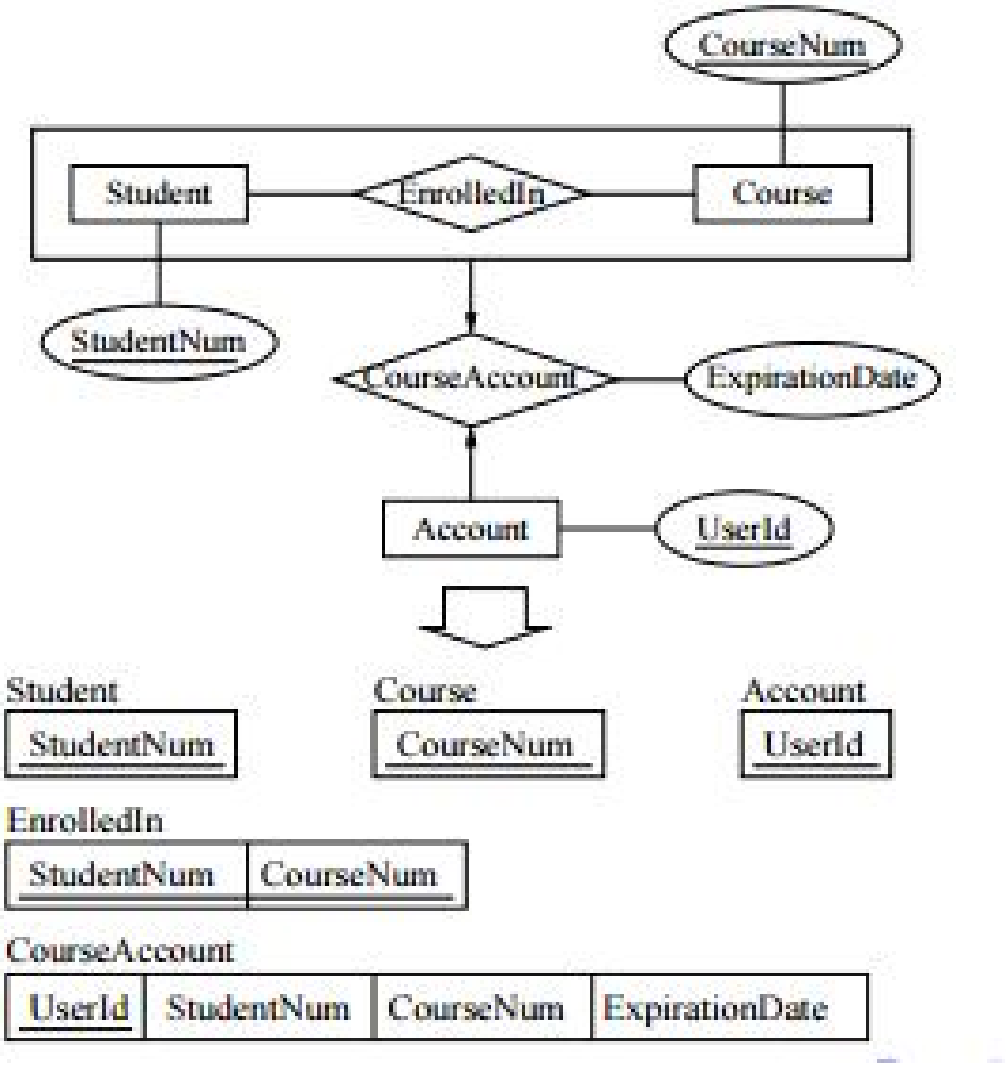
There are two different methods for transforming generalization to tabular form.

- Create a table for the generalized entity set. For each specialized entity set, create a table that includes a column for each of the attributes of that entity set plus a column for the primary key of the generalized entity set. Thus, for the E-R diagram shown below, we have three tables:
  - *account*, with attributes *account-number* and *balance*
  - *savings-account*, with attributes *account-number* and *interest-rate*
  - *checking-account*, with attributes *account-number* and *overdraft-amount*
- An alternative representation is possible, if the generalization is disjoint and complete. Then, we have two tables.
  - *savings-account*, with attributes *account-number*, *balance*, and *interest-rate*
  - *checking-account*, with attributes *account-number*, *balance*, and *overdraft-amount*



- **Conversion of Aggregation**

Conversion of aggregation into tables is illustrated as shown below;



UNIT-I
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Database applications were built directly on top of file system to overcome the following drawbacks of using file-systems [ ]

   (a) Data redundancy and inconsistency

   (b) Difficulty in accessing data

   (c) Data isolation

   (d) Integrity problems

   (A) (a) (B) (a) and (d)

   (C) (a), (b) and (c) (D) (a), (b), (c) and (d)

2. Data model is a collection of conceptual tools for describing [ ]

   (A) Data and data relationships (B) Semantics

   (C) Consistency constraints (D) All of the above

3. Identify the correct statement related to Physical level Abstraction:
   (A) describes how a record is stored [ ]
   (B) describes how schema is stored in a data base
   (C) hides details of data types
   (D) describes the data relationships in a data base

4. Choose the most appropriate choice with respect to conceptual design.

(A) Conceptual design is a documentation technique. Once the relation schemes are defined one can draw E-R diagrams from the relation schemes for documentation [ ]

(B) Conceptual design needs data volumn and processing frequencies to determine the size of the database

(C) Output of any conceptual design is an E-R diagram

(D) Conceptual design involves modelling the data requirements independent of the DBMS, operating system and the hardware.

5. An ER Model includes

   I. An ER diagram portraying entity types.

   II. Attributes for each entity type

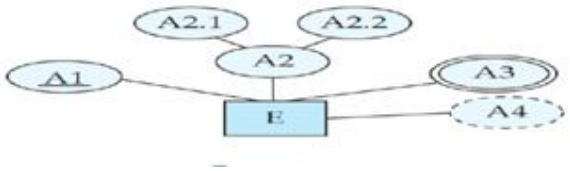
   III. Relationships among entity types.

   IV. Semantic integrity constraints that reflects the business rules about data not captured in the ER diagram. [ ]

(A) I, II, III & IV (B) I&IV

(C) I, II & IV (D) I & III

6. Refer the diagram below, where the attributes of relation E are characterized [ ]



(A) A4 is weak attribute
(B) A3 is multi valued attribute
(C) A2 is a derived attribute
(D) A3 is a foreign key attribute)

7. For a weak entity set to be meaningful, it must be associated with another entity set in combination with some of their attribute values is called as: [ ]

(A) Neighbour set (B) Strong entity set

(C) Owner entity set (D) Weak entity set

8. Which of the following statement is FALSE about weak entity set?

(A)Weak entities can be deleted automatically when their strong entity is deleted. [ ]

(B)Weak entity set avoids the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.

(C) A weak entity set has no primary key unless attributes of the strong entity set on which it depends are included.

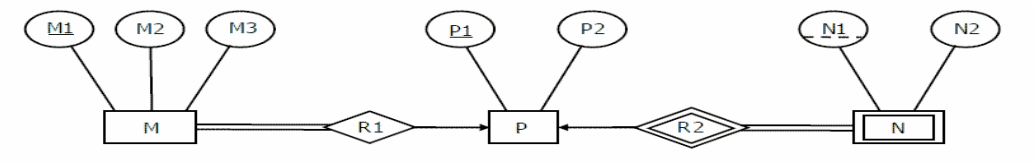
D) Tuples in a weak entity set are not partitioned according to their relationship with tuples in a strong entity set.

9. Every weak entity set can be converted into a strong entity set by:

(A) Using generalization [ ]
(B) Adding appropriate attributes
(C) Using aggregation

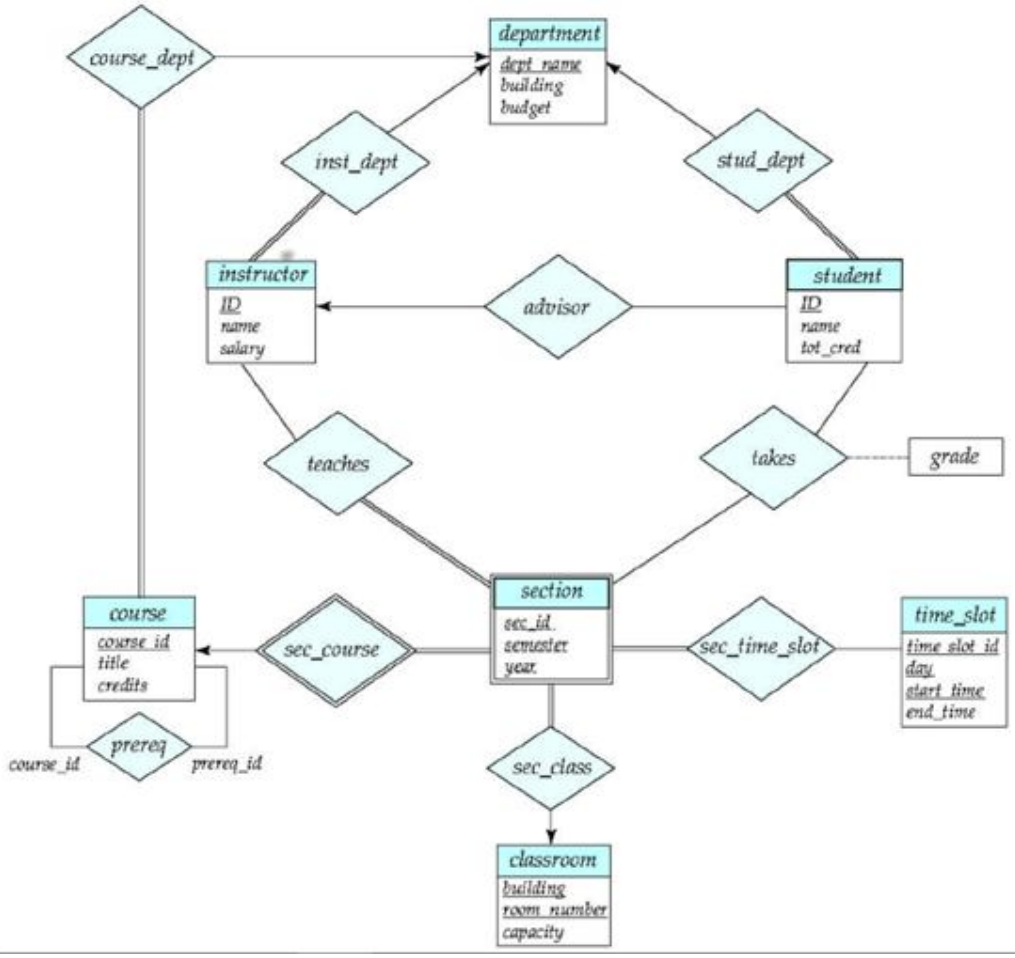
(D) None of the above

10. Consider the following ER diagram. The minimum number of tables needed to represent M, N, P, R1, R2 is [ ]



(A) 2 (B) 3 (C) 4 (D) 5

11.Consider the ER diagram given below:



Identify the correct statement(s) based on the ER model of the university. [ ]

(A) inst_dept is a relation set connects weak entity instructor with department.

(B) department is a strong entity.
(C) (time_slot_id, day) is the primary key for time_slot entity.
(D) course_id, prereq_id are the fields of prereq relation set.

12. Given the basic ER and relational models, Which of the following is INCORRECT? [ ]

(A) An attribute of an entity can have more than one value
(B) An attribute of an entity can be composite
(C) In a row of a relational table, an attribute can have more than one value
(D) In a row of a relational table, an attribute can have exactly one value or a NULL value

SECTION-B

SUBJECTIVE QUESTIONS

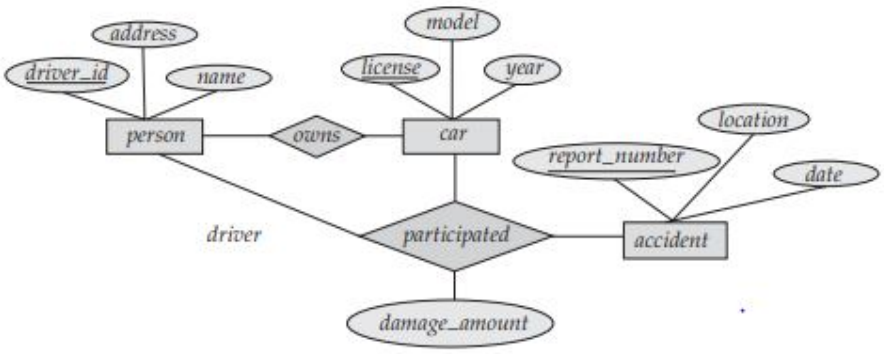
1) Define DBMS and explain why would choose a database system instead of simply storing data in operating system files?
2) What is a data model? Explain different types of data models with an example.
3) Briefly explain schema and instance with suitable example.
4) Describe purpose of ER diagrams and describe how entity, entity sets, Relationships, relationship sets are represented with an example.
5) What are the different types of attributes and keys used in ER model?
6) Quote suitable example to represent a strong and weak entity set through ER diagram.
7) Distinguish between weak entity set and strong entity set.
8) Outline the importance of EER modelling specialization and generalization with an example.
9) Illustrate with an example how to generate a relational-database schema from an ER model.
10) Illustrate with an example translation of relationship sets with participation constraints of an ER diagram to relational model.
11) Design a database for banking enterprise using ER Model.
12) Construct an ER diagram for college admission office section. The office maintains data about each class, including the instructor, the enrolment

and the time and place of the class meetings. For each student class pair a grade is recorded. Determine the entities and relationships.

13) Design a university level database for maintaining the student details of different colleges in the university. Only consider the personal details and the college and branch details of the student belong. represent the same using an ER diagram.

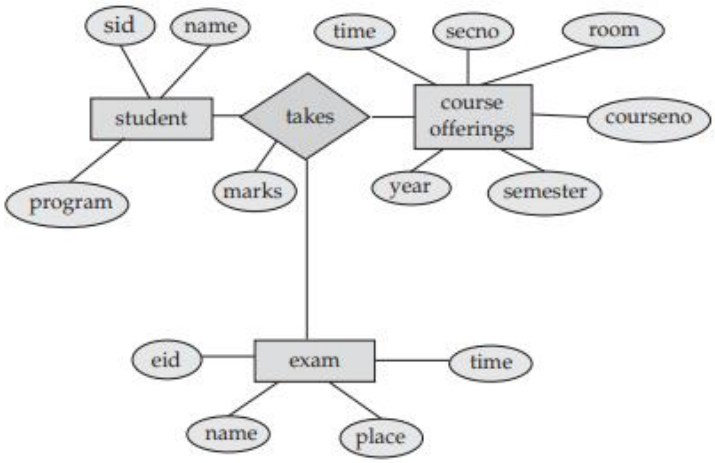
14) A company database needs to store information about employees(ssn, name, designation, salary, address, phone), departments(dno, dname, budget) and children of employees(name, age). Employee works in departments; each department is managed by an employee, a child must be identified uniquely by name when the parent(who is an employee; assume that only one parent works for the company) is known. We are not interested in information about child once the parent leaves the organization. Draw an ER diagram that captures this information.

15) Convert the following ER diagram into Relational tables.



E-R diagram for a car insurance company.

16) Convert the following ER diagram into Relational tables.

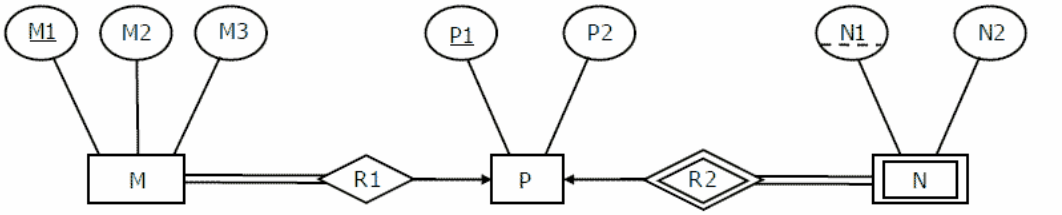


E-R diagram for marks database.

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Which of the following is a correct attribute set for one of the tables generated from below ER diagram. [ ]



a) {M1, M2, M3, P1}

b) {M1, P1, N1, N2}

c) {M1, P1, N1}

d) {M1, P1}

2. Let E1 and E2 be two entities in an E/R diagram with simple single-valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one-to-many and R2 is many-to-many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model? [ ]

a) 2 b) 3

b) 4 d) 5

UNIT – II

Relational Model& SQL

Relational model:Basic concepts, Schema and instances, Keys, Relational Algebra, SQL: DDL, DML, Integrity constraints, Defining different constraints on a table, Set operations, Aggregate Functions, Group by and Having clauses, Nested queries.

1. Relational Model Basic Concepts:

- In relational model, data is represented as a collection of ***tables*** also known as a ***relations***.
- Databases created using relational model are known as ***relational databases***.
- Database management system that operate on relational database is known as Relational Database Management System(RDBMS)
- Popular Relational Database Management Systems (RDBMS) are Oracle, DB2, SQLServer, Informix, MySQL, MSAccess etc.
- A relational database consists of a collection of tables, each of which is assigned a unique name.
- The degree, also called arity, of a relation is the number of fields in it.
- The cardinality of a relation is the number of tuples in it.

- Example:the following figure shows tables with unique names

customer_id	customer_name	customer_street	customer_city
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	balance
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

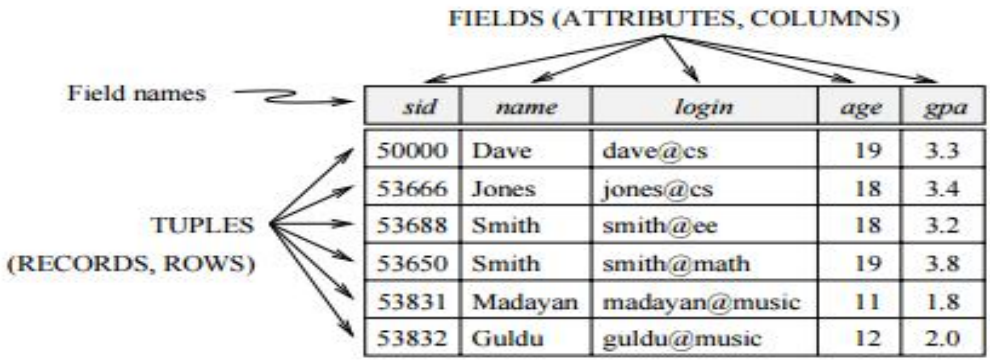
(b) The *account* table

customer_id	account_number
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Fig: tables with unique names

- Each table is described by a set of columns (also called as attributes, fields)
- For example, the following figure shows ***student*** table, which stores information about students.



sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure: Student table

- The ***student*** table has five columns: ***sid, name, login, age,*** and ***gpa.***
- Each row of this table records information about a student.
- **Relation schema:** A relation schema consists of a list of attributes and their corresponding domains.
- **Domain** of an attribute represents the set of permitted values of an attribute.
- For example, the domain of the name attribute of student relation is the set of all possible student names.
- Thus, the relation schema of the student relation can be written as
- student(sid:integer, name:string, login:string, age:integer, gpa:float)
- **Relation instance** – refers to a specific instance of a relation, i.e. containing a specific set of rows.
- For example, the instance of ***student relation*** has 6 rows corresponding to 6 students.
- **Relation Cardinality -** is the number of tuples or rows in the relation.
- For example, cardinality of student relation is 6 as it has 6 rows.
- **Relation Degree or Arity**- is the number of attributes (or columns) in the relation.
- For example, degree or arity of student relation is 5 as it has 5 columns.

NPTEL

- Relational Database Management System- A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by

- E. F. Codd. A Relational database comprises of various relations or tables. We will start with the basic concepts of RDBMS which are as follows:

l) Table- The data in an RDBMS is stored in database objects which are called as relations or tables. This table is basically a collection of related data entries and it consists of numerous columns and rows. Below mentioned table shows a student table in a relationaldatabase.

S_ID	S_NAME	S_DOB	S_ADDRESS
001	Ajay	13-OCT-1991	Jaipur
002	Bhanu	02-JAN-1992	Delhi
003	Devika	04-JUL-1990	Kota

2) Field- Every table is broken up into smaller entities called fields or attributes. The fields in the STUDENT table consist of S_ID, S_NAME, S_DOB and S_ADDRESS. A field is a column in a table that is designed to maintain specific information about every record in the table.

3) Record- A record, also called as a row or tuple, is each individual entry that exists in a table. For example, there are 3 records in the above STUDENT table and one record specifies the complete information of an entity.

4) Column- A column is a vertical entity in a table that contains all information associated with a specific field in atable.

5) NULL value- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a recordcreation.

6) Database Schema- A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

7) Database Instance- A database instance is a state of operational database with data at any given point of time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

2. Schema and Instances

- Databases change over time.
- The information in a database at a particular point in time is called an instance of the database.

- The overall design of the database is called the database schema. In other words, a database schema consists of collection of relational schemas.
- The overall structure of the relation (or table) is known as relation schema. E.g. student relation schema is shown as
  - student (sid, name, login, age, gpa)
- The information in a relation (or table) at a particular point in time is called an instance of the relation. E.g. figure 1.7 shows instance of the *student relation.*

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Fig. : Instance of *student relation*

3. Keys

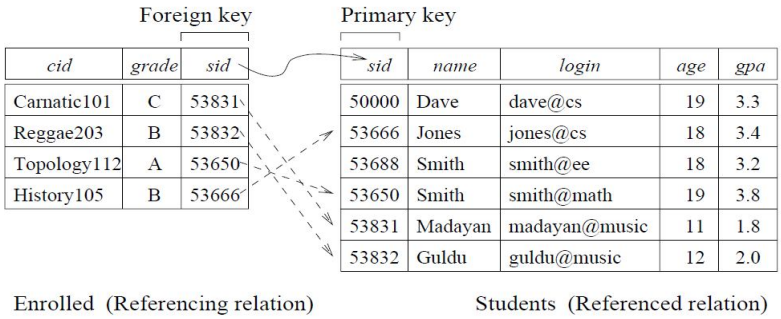
1. **Super Key** – set of one or more attributes that uniquely identifies a tuple in a relation is called as a super key.
2. **Candidate Key** – minimal set of attributes that uniquely identifies a tuple in a relation is called as a candidate key.
3. **Primary Key** – is a candidate key which uniquely identifies a tuple in a relation. The two properties of primary key are unique and not null.
4. **Foreign Key –**Ensure the referential integrity of the data in one table to match values in another table. Ensure that the foreign key in the child table must match with the primary key in the parent table.

Example:

```
CREATE TABLEStudent (sidnumber(3) UNIQUE,
                     namevarchar(60) NOT NULL,
                     loginvarchar(20),
                     Age number(3),
                     gpa number(2,2));

CREATE TABLEEnrolled (sidnumber(3),
                      cidvarchar(20),foreign key(sid) references student)
```

- The following figure shows the mapping mapping of Foreign key with Primary key.



CHILD TABLE PARENT TABLE

Fig: mapping of Foreign key with Primary key

NPTEL

Key-

- A key is a single or combination of multiple fields in a table. It is used to fetch or retrieve record(s) from a table according to the condition/requirement.
- Keys are also used to create relationship among different database tables or views.
- We have following types of keys in SQL which are used to fetch records from tables and to make relationship among tables or views.

1) **Super Key**-Super Key is the combination of attributes that can be used to identify a record uniquely in a table and is a set of one or more than one keys. Primary key, Unique key, Alternate key are subset of SuperKeys.
2) **Candidate Key**-A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table. There can be multiple candidate keys in one table. Each candidate key can work as primarykey.
3) **Primary Key**- Primary Key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It cannot accept null and duplicate values. Only one candidate key can be primarykey.
4) **Alternate Key**- An Alternate Key is a key that can be work as a primary key. Basically it isa candidate key that is not a primarykey.
5) **Composite/Compound Key-** Composite Key is a combination of more than one fields/columns of a table. It can be a candidate key or primarykey.
6) **Unique Key-** Unique Key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It is like Primary key but it can accept only one null value and it cannot have duplicatevalues.
7) **Foreign Key-** Foreign Key is a field in a database table that is primary key in another table. It is used to link two tables in a databaseand can accept multiple null, duplicatevalues.

4. Relational Algebra (NPTEL)

- Relational algebra consists of simple and powerful operators to construct new relation(s) from given relations.
- It is part of first order logic and algebra of sets. If relations are stored data, then constructed relations can be taken as answers to queries on stored data.
- It deals with a set of finite relations which is closed under certain operators.
- Relational algebra was proposed by T Codd as an algebra on sets of tuples (i.e., relations) and is restricted the operands of relational algebra to finite relations only.
- Relational Algebra and relational calculus are two formal query languages for relational model and were developed prior to the development of commercial query language Structured Query Language (SQL).
- Relational algebra consists of operators and atomic operands.
  - The atomic operands in relational algebra are
  - Variables that stand relations and
  - constants, termed as finite relations.
  - All operands and results of expressions are sets.
  - The operators in relational algebra are broadly classified into four categories, namely,
  - Set operators,
  - Unary operators to remove parts of a relation,
  - Operators to combine the tuples of two relations and
  - Rename operator used to change relation schema without affecting the tuples of a relation.
- The basic set of operators of relational algebra enable the user to specify basic retrieval requests as relational algebra expressions.
- There are two types of operators, namely, Unary operators (selection and projection) and binary operators (cross product, union and set difference).
- Each operator considers one or two relation(s) as input and

produces new relation as output.

- Relational algebra expression is a sequence of relational algebra operators on relation(s) and the output produced by relational algebra expression is termed as result of database query.

- This language is accepted as formal foundation for relational model and is used as a basis for implementing and optimizing queries in the query processing and query optimization modules of DBMS.

- Relational algebra expression is also represented in the form of query tree where leaf nodes are relations and intermediate nodes are operators and finally the root node produces final answer for user query.

- Basic Operations:

  - selection (σ),
  - Projection (∏),
  - Union (∪) ,
  - Set difference (-) and
  - Cartesian Product (X)

Operation	Purpose	Notation
Selection	Selects all tuples that satisfy the selection condition from a relation R	σp(R)
Projection	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	∏A1, A2, An (R) where A1, A2 , An are attribute names of
Union	Produces all combinations of tuples from R1 and R2 that satisfy the join condition	R1 ∪ R2
Set Difference	Produces a relation that includes all tuples in R1 that are not in R2 and R2 must be union compatible	R1 - R2
Cartesian Product	ProducesarelationthathastheattributesofR1 and R2 and includes as tuples all possiblecombinations of tuples from R1 andR2	R1 X R2

4.1 Additional RelationalOperators:

- There are several additional operators and whichcanbederivedfrombasicoperators, such as
  - Set Intersection
  - Division
  - Natural join
  - semi join
  - antijoin

  - **SetIntersection(R1∩R2):**Itgivesthesetthatcontainsalltuplesof R1 thatalsobelongto R2(orequivalently,alltuplesofR2thatalsobelongtoR1),butnoothert uples.Settheoretic notationforR1∩R2={x:x□R1andx□R2}.

  - **Division (R1 ÷ R2):** Division between relations R1 and R2 results in the restrictions of tuplesinR1totheattributenamesuniquetoR1,i.e.,intheheaderofR1 butnotintheheader ofR2,forwhichitholdsthatalltheircombinationswithtuplesinR2are presentinR1.

  - **Natural Join (R1⋈R2):** Natural join (⋈) operator considers two relations R1 and R2 as relationsandproducesthesetofallcombinationsoftuplesinR1andR 2thatareequalon their common attribute names (or columnnames).

  - **SemiJoin(R1⋉R2):**Thisissimilartonaturaljoinandtheresultyielde dbythisisonlythe setofalltuplesinfirstrelationforwhichthereisatupleinsecondrelatio nthatisequalon their common attribute names.
  - **Antijoin:(R1▷R2):**Itisconverseofsemijoinoperation.Itreturnsalltu plesintheresult ofexpressionR1suchthattherearenotuplesintheresultofR2withma tchingvaluesforthe sharedattributes.

1. SQL:

- SQL stands for “Structured Query Language.” It is developed by IBM in 1974 and originally it is called as SEQUEL (Structured English QUEry Language).
- SQL commands consist of English-like statements which are used to query, insert, update, and delete data.

- SQL is a "nonprocedural" or "declarative" language. Here nonprocedural means that, when we want to retrieve data from the database it is enough to tell SQL what data to be retrieved, rather than how to retrieve it. The DBMS will take care of locating the information in the database.
- ANSI and ISO standard SQL:
    - SQL-86
    - SQL-89
    - SQL-92
    - SQL:1999 (language name became Y2K compliant!)
    - SQL:2003
- The first commercial DBMS that supported SQL was Oracle i.n 1979

SQL Commands: the following figure shows different SQL Commands

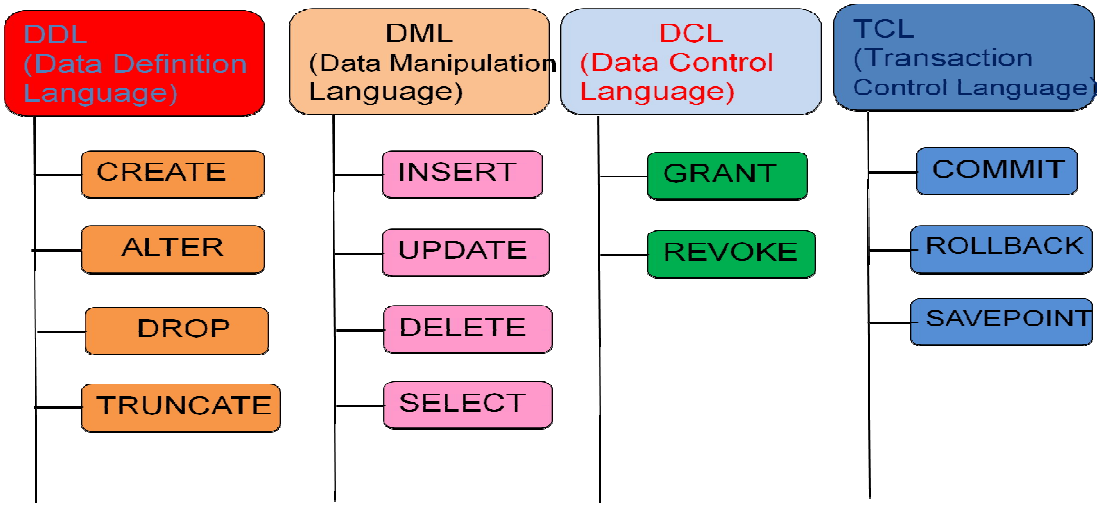


Figure: SQLCommands

1. **DDL commands -** are used to define a database, including creating, altering, and dropping tables and establishing constraints.

2. **DML commands** - are used to maintain and query a database, including updating, inserting, modifying, and querying data.

3. **DCL commands** - are used to control a database including administering privileges and saving of data. DCL commands are used to determine whether a user is allowed to carry out a particular operation or not.

4. **TCL commands -** are used to manage transactions in database. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

SQL Commands

- **Create Table Command:**

Is used to define a database, including creating, altering, and dropping tables and establishing constraints.

Syntax:

CREATE TABLE *table name*
(column-name1 data-type-1 [constraint],
column-name2 data-type-2 [constraint],
column-nameN data-type-N [constraint]);

Example

create table *branch*
(*branch_name* **varhar**(15)**,**
branch_city **varhar(**30),
assets **integer**);

- **Domain Types in SQL: (NPTEL)**

1. **char(n).** Fixed length character string, with user-specified length *n.*
2. **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
3. **int.** Integer (a finite subset of the integers that is machine-dependent).
4. **smallint.** Small integer (a machine-dependent subset of the integer domain type).
5. **number(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
6. **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
7. **float(n).** Floating point number, with user-specified precision of at least *n* digits.

SQL Data Types-Data storage format of an object is defined by SQL data types. These objects can be a variable or columns or expressions and data

can be numeric, character, string, binary, date and time etc.

1) Numeric Data Types inSQL:

- INTEGER – Stores integer number, values from -2,147,483,648 to2,147,483,647.
- SMALLINT – Stores small integer number, values from -32,768 to32,767.
- TINYINT – Stores tiny integer number, values from 0 to255.
- NUMERIC(P,S) – Stores values from -10^38 +1 to 10^38 -1. Where 'p' isprecision value and 's' is scalevalue.
- REAL – Single precision floating point number, stores values from -3.40E + 38 to3.40E
- + 38.
- DECIMAL(P,S) – Stores values from -10^38 +1 to 10^38 -1, where 'p' is precisionvalue and 's' is scalevalue.
- FLOAT(P) – Stores values from -1.79E + 308 to 1.79E + 308, where 'p' isprecision value.
- DOUBLE PRECISION – Double precision floating pointnumber.
- BIT(X) – Where 'x' is the number of bits to store, stores value from 0 to1.
- BIT VARYING(X) – 'X' is the number of bits to store (length can vary up tox).

2) Character DataTypes:

- CHAR(X) – Where 'x' is the character's number to store. It can store upto 8,000 characters.
- VARCHAR2(X) – Where 'x' is the character's number to store. It can store upto8,000 characters.
- VARCHAR(MAX) – It can store upto 231characters.
- text – Can store upto 2,147,483,647characters.

3) Binary Data Types inSQL:

- binary – Can store upto 8,000 bytes (Fixed-length binarydata).
- varbinary – Can store upto 8,000 bytes. (Variable length binarydata).
- varbinary(max) – Can store upto 231 bytes (Variable length Binarydata).
- image – Can store upto 2,147,483,647 bytes. (Variable length BinaryData).

4) Date and Time Data Types inSQL:

- DATE – Stores month, days and yearvalues.
- TIME – Stores hour, minute and secondvalues.
- TIMESTAMP – It stores year, month, day, hour, minute and second values.

SQL Operators- The symbols which are used to perform logical and

mathematical operations in SQL are called SQL operators. Three types of operators used in SQL are as follows:

1) ArithmeticOperators:

Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in SQL statements.

Arithmetic Operators	Example/Description
+ (Addition)	A+B
– (Subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

2) RelationalOperators:

Relational operators in SQL are used to find the relation between two columns. i.e. to compare the values of two columns in SQL statements.

Relational Operators	Example/Description
>	x > y (x is greater than y)
<	x < y (x is less than y)
>=	x >= y (x is greater than or equal to y)
<=	x <= y (x is less than or equal to y)
=	x <= y (x is less than or equal to y)
!= or <>	x != y or x <> y (x is not equal to y)
!<	x !< y (x is not less than y)
!>	x !> y (x is not greater than y)

3. LogicalOperators:

Logical operators in SQL are used to perform logical operations on the given expressions in SQL statements. There are many operators in SQL which are used in SQL statements in the WHERE clause. They are,

- AND
- OR
- NOT
- BETWEEN…AND
- IS NULL, IS NOTNULL
- LIKE
- UNIQUE

➢ **Basic Insertion and Deletion of Tuples:**

- Newly created table is empty
- Add a new tuple to *account*

insert into *account*

values('A-9732', 'Perryridge', 1200);

Note: Insertion fails if any integrity constraint is violated

- Delete *all* tuples from *account*

  delete from *account*

  Note: Will see later how to delete selected tuples

➢ **Drop and Alter Table Constructs:**

- The **drop table** command deletes all information about the dropped relation from the database.
- The **alter table** command is used to add attributes to an existing relation:

  alter table *r* **add** *A D*

  - where*A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
  - All tuples in the relation are assigned *null* as the value for the new attribute.
- The **alter table** command can also be used to drop attributes of a relation:

  alter table *r* **drop** *A*

  - where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases

Tables: consider the following figure for writing queries

branch_name	*branch_city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

branch relation

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

loan relation

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

customer relation

Basic Query Structure

A typical SQL query has the form:

select A_1, A_2, ...,A_n
from r_1, r_2, ..., r_m
where P

- A_i represents an attribute
- R_i represents a relation
- P is a predicate.

The result of an SQL query is a relation.

The select Clause:

- The **select** clause list the attributes desired in the result of a query
- Example: find the names of all branches in the *loan* relation:

select *branch_name* **from** *loan;*

branch_name
Round Hill
Downtown
Perryridge
Perryridge
Downtown
Redwood
Mianus

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  Example: *Branch_Name* ≡ *BRANCH_NAME* ≡ *branch_name*
  Some people use upper case wherever we use bold font.

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select**.**
- Find the names of all branches in the *loan* relation, and remove duplicates

select distinct *branch_name***from** *loan;*

Branch_name
Round Hill
Downtown
Perryridge
Redwood
Mianus

- The keyword **all** specifies that duplicates not be removed.

select all*branch_name***from** *loan;*

- An asterisk in the select clause denotes "all attributes"

select*****from** *loan*

- The **select** clause can contain arithmetic expressions involving the operation, +, –, □, and /, operating on constants or attributes of tuples.

Example: **select***loan_number, branch_name,*
amount *100
from*loan*

The where Clause:

- The **where** clause specifies conditions that the result must satisfy.
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

select*loan_number*
from *loan*
where *branch_name* ='Perryridge'**and** *amount* > 1200

- Comparison results can be combined using the logical connectives **and, or,** and **not.**

The from Clause:

- The **from** clause lists the relations involved in the query
- Find the Cartesian product *borrower X loan*

select *from *borrower, loan;*

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

select*customer_name, borrower.loan_number, amount*
from *borrower, loan*

where *borrower.loan_number = loan.loan_number* **and** *branch_name* = 'Perryridge'

The Rename Operation:

- SQL allows renaming relations and attributes using the **as** clause: *old-name* **as** *new-name*
- Example: Find the name, loan number and loan amount of all customers; rename the column name *loan_number* as *loan_id.*

select *customer_name, borrower.loan_number* **as** *loan_id, amount*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number*

Tuple Variables:

- Tuple variables are defined in the **from** clause.
- Find the customer names and their loan numbers and amount for all customers having a loan at some branch

select *customer_name, T.loan_number, S.amount*
from *borrower T, loan S*
where *T.loan_number = S.loan_number*

- Find the names of all branches that have greater assets than some branch located in Brooklyn.

select distinct *T.branch_name*
from *branch T, branch S*
where *T.assets>S.assets* **and** *S.branch_city* = 'Brooklyn'

String Operations:

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - **Percent sign %** : represents zero, one or more than one character.
  - **Underscore sign** _ : represents only one character.
- Find the names of all customers whose street includes the substring "Main".

select *customer_name*

from *customer*

where *customer_street* **like** '% Main%'

SQL supports a variety of string operations such as

- concatenation (using "||")
- converting from upper to lower case (and vice versa)
- finding string length, extracting substrings, etc.

Ordering the Display of Tuples:

- List in alphabetical order the names of all customers having a loan in Perryridge branch

select distinct *customer_name*

from *borrower, loan*

where *borrower loan_number = loan.loan_number* **and** *branch_name* = 'Perryridge'

order by *customer_name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.

Example: **order by** *customer_name* **desc**

8. Integrity Constraints :

5. **CHECK -** Ensures that the value in a column meets a specific condition. E.g. check(account_balance>0).

   Example:

   CREATE TABLE *Student* (sidnumber(3) NOT NULL,
   Name varchar(60) NOT NULL,
   Age number(3),
   CHECK(sid> 0));

6. **NOT NULL** - Indicates that a column cannot store NULL value. E.g. Account_numberchar(10) not null.

Example:

CREATE TABLE*Student* (sidnumber(3) **NOT NULL**,
Name varchar(60) **NOT NULL**,
Age number(3));

7. **UNIQUE** - The UNIQUE constraint imposes that every value in a column or set of columns be unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns. E.g. UNIQUE(Name, DOB).

Example:

CREATE TABLE*Student* (sidnumber(3) **UNIQUE**,
Name varchar(60) NOT NULL,
Age number(3));

8. **FOREIGN KEY** – Ensure the referential integrity of the data in one table to match values in another table. Ensure that the foreign key in the child table must match with the primary key in the parent table.

Example:

CREATE TABLE*Student* (sidnumber(3) **UNIQUE**,
namevarchar(60) NOT NULL,
loginvarchar(20),
Age number(3),
gpa number(2,2));

CREATE TABLE*Enrolled* (sidnumber(3),
cidvarchar(20),
foreign key(sid) references student)

- The following figure shows the mapping mapping of Foreign key with Primary key.

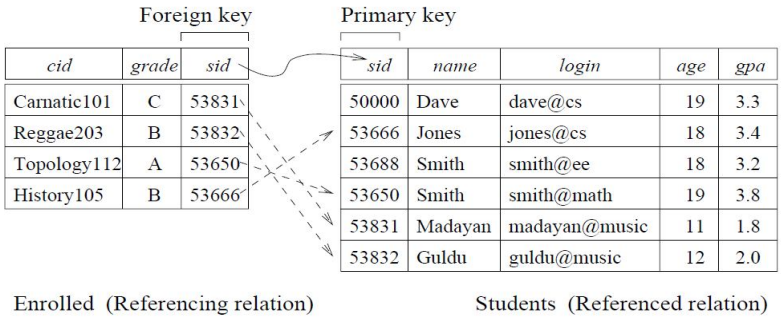


Fig: mapping of Foreign key with Primary key

- If we try to insert the *tuple (55555, Art104, A)* into **Enrolled** table, then the foreign key constraint is violated because there is no tuple in Students table with sid 55555 and so this insertion is rejected.

ON DELETE CASCADE

- When the clause ON DELETE CASCADE is included in the child table, and if a row is deleted from the parent table then the corresponding referenced value in the child table will also be deleted.

ON DELETE SET NULL

- If ON DELETE SET NULL clause is included in the child table means, whenever a row in the parent table is deleted, then the corresponding referenced value in the child table will be set null.

- **Difference between UNIQUE and NOTNULL Constraint**

NOTNULL Constraint	UNIQUE Constraint
An attribute declared as NOTNULL will not accept NULL values	An attribute declared as UNIQUE can accept NULL values
An attribute declared as NOTNULL will accept duplicate values	An attribute declared as UNIQUE will not accept duplicate values

- **Difference between UNIQUE and PRIMARY KEY Constraint**

PRIMARY KEY constraint	UNIQUE constraint
an attribute declared as primary key will not accept NULL values	an attribute declared as UNIQUE will accept NULL values
only one PRIMARY KEY can be defined for each table	more than one UNIQUE constraint can be defined for each table

GROUP BY: (NPTEL)

- The GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.
- Sometimes it is required to get information not about each row, but about each group.
- Example: consider the Customer_Loan table that has data about all the loans taken by all the customers of the bank. Assume that we want to retrieve the total loan-amount of all loans taken by each customer.
- Related rows can be grouped together by the GROUP BY clause by specifying a column as a grouping column.
- In the above example, the Cust_ID will be the grouping column.
- In the output table all the rows with an identical value in the grouping column will be grouped together. Hence, the number of rows in the utput is equal to the number of distinct values of the grouping column.

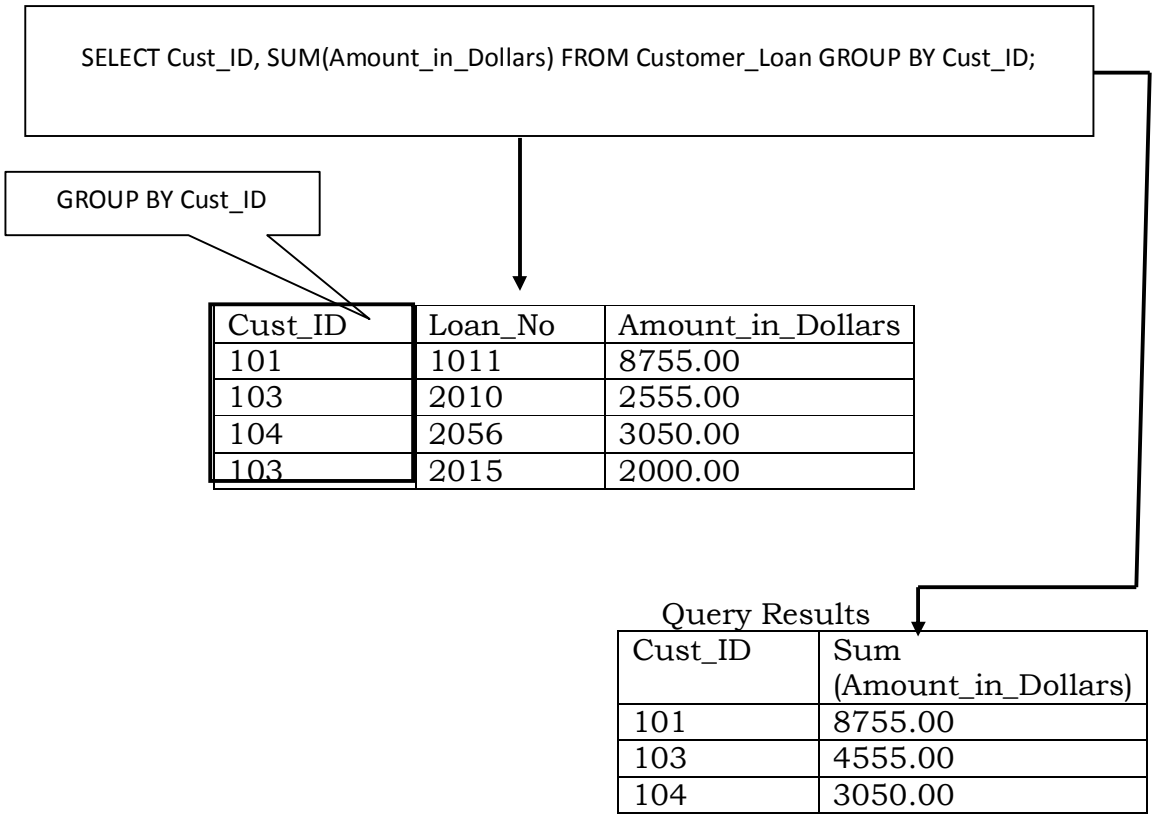
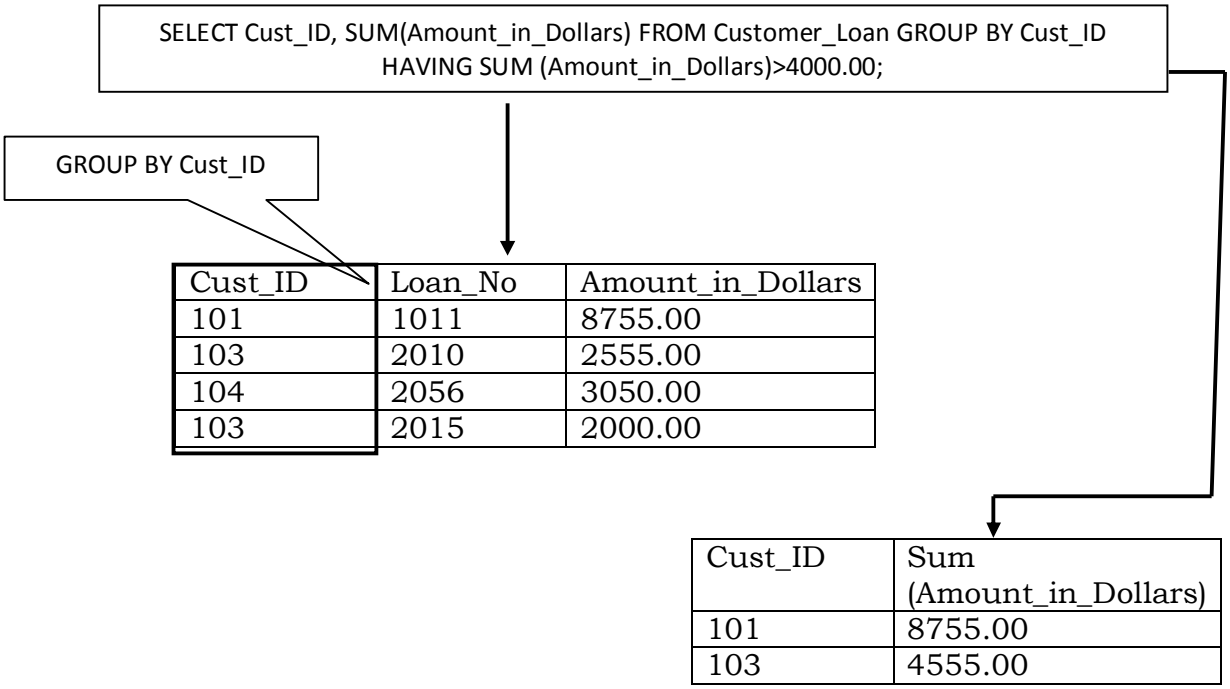


Figure: Example of Group BY Clause

HAVING: (NPTEL)

- The HAVING clause is used along with the GROUP BY clause. The HAVING clause can be used to select and reject row groups.
- The format of the HAVING clause is similar to the WHERE clause, consisting of the keyword HAVING followed by a search condition.
- The HAVING clause thus specifies a search condition for groups.



Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Cust_ID	Sum (Amount_in_Dollars)
101	8755.00
103	4555.00

Figure: Example of HAVING Clause

2. Set Operations in SQL:

SQL provides 3 types of set operations.

1. **UNION** – Let R and S are two union compatible relations. Then the UNION operation returns the tuples that occur either in R or S or both.
   - Two relation instances are said to be union-compatible if the following conditions hold:
   - they have the same number of the columns, and
   - Corresponding columns, taken in order from left to right, have the same *data types.*
2. **INTERSECT** – Let R and S are two union compatible relations. Then the INTERSECT operation returns the tuples that are common to the two relations.
3. **MINUS** – If R and S are two union compatible relations then **R MINUS** S returns the tuples that are present in R but not in S.

Examples using set operations in SQL:

Consider the following two relations.

Hard_Disk	Speed	OS
100GB	1.8 GHz	Linux
200GB	2.0GHz	Windows
250GB	2.8GHz	Windows

IBM_Desktop

Hard_Disk	Speed	OS
100GB	1.8 GHz	Linux
200GB	2.0GHz	Windows
250GB	2.0GHz	Windows

Dell_Desktop

```
SELECT * FROM IBM_Desktop
UNION
SELECT * FROM Dell_Desktop;
```

Hard_Disk	Speed	OS
100GB	1.8 GHz	Linux
200GB	2.0GHz	Windows
250GB	2.8GHz	Windows
250GB	2.0GHz	Windows

```
SELECT * FROM IBM_Desktop
UNION ALL
SELECT * FROM Dell_Desktop;
```

Hard_Disk	Speed	OS
100GB	1.8 GHz	Linux
100GB	1.8 GHz	Linux
200GB	2.0GHz	Windows
200GB	2.0GHz	Windows
250GB	2.8GHz	Windows
250GB	2.0GHz	Windows

```
SELECT * FROM IBM_Desktop
      INTERSECT
SELECT * FROM Dell_Desktop;
```

Hard_Disk	Speed	OS
100GB	1.8 GHz	Linux
200GB	2.0GHz	Windows

```
SELECT * FROM IBM_Desktop
MINUS
SELECT * FROM Dell_Desktop;
```

Hard_Disk	Speed	OS
250GB	2.8GHz	Windows

Consider sailors, boats, reserves tables for set operations:

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Boats

- Find the names of sailors who have reserved a red or a green boat.

```
SELECT S.sname
  FROM Sailors S, Reserves R, Boats B
  WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
  UNION
  SELECT S.sname
  FROM Sailors S, Reserves R, Boats B
  WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'green';
```

sname
Dustin
Lubber
Horatio

- Find the sids of sailors who have reserved both red and green boats.

```
SELECT S.sid
```

```
FROM Reserves R, Boats B
WHERE R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S.sid
FROM Reserves R, Boats B
WHERE R.bid = B.bid AND B.color = 'green';
```

sid
22
31

- Find the sids of sailors who have reserved red boats but not green boats.

```
SELECT S.sid
  FROM Reserves R, Boats B
  WHERE R.bid = B.bid AND B.color = 'red'
  MINUS
  SELECT S.sid
  FROM Reserves R, Boats B
  WHERE R.bid = B.bid AND B.color = 'green';
```

sid
64

- *Find the sids of sailors who have reserved red boats but not green boats.*

```
SELECT S.sid
  FROM Reserves R, Boats B
  WHERE R.bid = B.bid AND B.color = 'red'
  MINUS
  SELECT S.sid
  FROM Reserves R, Boats B
  WHERE R.bid = B.bid AND B.color = 'green';
```

sid
64

3. Aggregate Functions in SQL:

1. MIN() - Returns the smallest value in a given column
2. MAX() - Returns the largest value in a given column
3. SUM() - Returns the sum of numeric values in a given column
4. AVG() - Returns the average value of a given column

5. COUNT() - Returns the total number of values (excluding NULL values) in a given column
6. COUNT(*) - Returns the number of rows in a table (including NULL values).

EXAMPLES

- *Find the average age of all sailors.*

```
SELECT AVG(S.age) AS avgAge
  FROM Sailors S;
```

avgAge
37.4

- *Find the average age of sailors with a rating of 10.*

```
SELECT AVG(S.age) AS avgAgeOfRating10
  FROM Sailors S
  WHERE S.rating = 10;
```

avgAgeOfRating10
25.5

- *Find the age of the youngest sailor.*

```
SELECT MIN(S.age) AS youngestSailorAge
 FROM Sailors S;
```

youngestSailorAge
16.0

- *Find the age of the oldest sailor.*

```
SELECT MAX(S.age) AS oldestSailorAge
 FROM Sailors S;
```

oldestSailorAge
63.5

- *Find the total number of sailors.*

```
SELECT COUNT(S.sid) AS NoOfSailors
 FROM Sailors S;
```

NoOfSailors
10

- *Find the number of sailors with rating 10.*

```
SELECT COUNT(S.sid) AS NoOfSailorsWithRating10
 FROM Sailors S
 WHERE S.rating = 10;
```

NoOfSailorsWithRating10
10

- *Count the distinct ratings.*

```
SELECT COUNT(DISTINCT S.sid) AS NoOfRatings
 FROM Sailors S;
```

NoOfRatings
6

5. Nested Queries:

- A **nested query is a query** that has another query embedded within it. The embedded query is called a **sub query.**

- *Find the names of sailors who have reserved boat 103.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECTR.sid
                 FROM Reserves R
                 WHERE R.bid = 103 )
```

sname
Dustin
Lubber
Horatio

- *Find the names of sailors who have not reserved boat 103.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN ( SELECTR.sid
                     FROM Reserves R
                     WHERE R.bid = 103 )
```

sname
Horatio

- *Find the names of sailors who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN ( SELECTR.sid
                     FROM Reserves R
                     WHERE R.bid IN (SELECT R.bid
                          FROM Boats B
                          WHERE B.color='red');
```

sname
Dustin
Lubber
Horatio

UNIT-II

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. For every teacher record in a database, there is an attribute called Department. This attribute specifies the department name. At times, the name may contain the numeric department id concatenated with it. However, it can never comprise only of the department id. Department name is optional in a teacher record.

Identify the correct components for the domain of the attribute Department. [ ]

a) Date b) Integer c) NULL d) Alphanumeric (String and Integer)

2. Identify the correct statement(s). [ ]

a) A Candidate Key is a set of one or more attributes that, taken collectively, allows us to uniquely identify any entity in the entity set

b) A Candidate Key for which no proper subset is also a Candidate Key is called a Super Key

c) A Super Key is a set of one or more attributes that, taken collectively, allows us to uniquely identify any entity in the entity set

d) A Super Key for which no proper subset is also a Super Key is called a Candidate Key

i)a,b ii) a,c,d iii) a iv) c,d

3. Identify the valid data-types, which can be used in SQL to define the type of data. [ ]

a) Varchar b) string c) real d) float

i) a,b ii) c,d iii) a iv) a,c,d

4. Consider the course table.

course(course_id, title, dept_name, credits).

Create a new course 'HS-001', titled 'SUPW', with 10 credits for department 'HSC'.

Identify the appropriate SQL [ ]

a) Insert into table course values('HS-001','SUPW', 'HSC',10)

b) Insert into course ('HS-001','SUPW', 'HSC',10)

c) Insert into course values('HS-001','SUPW', 'HSC',10)

d) Insert into table course ('HS-001','SUPW', 'HSC',10)

5. The command to remove rows from a table 'CUSTOMER'is: [ ]

a) Remove From Customer ... b) Drop From Customer . ..

c) Delete From Customer Where .. . d) Update From Customer . .

6. The primary key must be [ ]

a) Not null b) Unique c) a or b d) Both a and b

7. The set of permitted values of each attribute is called [ ]

a) Domain b) Tuple c) Relation d) Schema

8. SQL Query to find an employee whose Salary is equal or greater than 10000 is_____

9. SQL Query to find name of employee whose name Start with 'M' is _____ .

10. Select _____ dept_name from Instructor, Here which of the following displays the unique values of the column? [ ]

a) All b) From c)Distinct d) Name

11. Select * from employee where salary>10000 and dept_id=101; Which of the following fields are displayed as output? [ ]

a) Salary, dept_id c) Employee

b) Salary d) All the field of employee relation

12. Which of the following statements contains an error? [ ]

a) Select * from emp where empid=10003;

b) Select empid from emp where empid = 10006;

c) Select empid from emp;

d) Select empid where empid = 1009 and lastname = 'GELLER';

13. The employee information in a company is stored in the relation

Employee (name, gender, salary, deptName)

Consider the following SQL query

Select deptName From Employee

Where gender = 'M' Group by deptName

Having avg(salary) > (select avg (salary) from Employee)

It returns the names of the department in which [ ]

(a) the average salary is more than the average salary in the company

(b) the average salary of male employees is more than the average salary of all male employees in the company

(c) the average salary of male employees is more than the average salary of employees in the same department.

(d) The average salary of male employees is more than the average salary in the company

14. The relation book (title, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list? [ ]

Select Title From book as B

Where (Select count(*)from book as T Where T.price>B.price) < 5)

a) Titles of the four most expensive books

b) Title of the fifth most inexpensive book

c) Title of the fifth most expensive book

d) Titles of the five most expensive books

SECTION-B

SUBJECTIVE QUESTIONS

1. What is a relational model and explain different DDL, DML commands in SQL with syntax.
2. Outline the basic structure of SQL with suitable examples.
3. Write in detail about different types of constraints that can be specified on a relation.
4. List and explain set operations and different aggregate functions in SQL.
5. Consider the following schema :

Emp(empid, emp_name, emp_sal,Date)

i. Query to find second highest salary of Employee.
ii. SQL Query to find Max Salary from each department.
iii. Write SQL Query to display the current date.
iv. Find all Employee records containing the word "Joe", regardless of whether it was stored as JOE, Joe, or joe.

6. Write the SQL expressions for the following relational database?

Sailor schema (sailor id, Boat id, sailorname, rating, age)

Reserves (Sailor id, Boat id, Day)

Boat Schema (boat id, Boatname, color)

i) Find the age of the youngest sailor for each rating level?

ii) Find the No.of reservations for each red boat?

iii)Find the average age of sailor for each rating level that at least 2 sailors.

7. Consider the following relational schema:

Emp(*eid:* integer, *ename:* string, *age:* integer, *sala111:* real)

Works(*eid:* integer, *did:* integer, *peLtime:* integer)

Dept(did: integer, *dname:* string, *budget:* real, *managerid:* integer)

i. Give an example of a foreign key constraint that involves the Dept relation. What are the options for enforcing this constraint when a user attempts to delete a Dept tuple?
ii. Write the SQL statements required to create the preceding relations, including appropriate versions of all primary and foreign key integrity constraints.
iii. Define the Dept relation in SQL so that every department is guaranteed to have a manager.
iv. Write an SQL statement to add John Doe as an employee with *eid*= 101, *age* = 32 and *salary* = 15,000.
v. Write an SQL statement to give every employee a 10 percent raise.
vi. Write an SQL statement to delete the Toy department. Given the referential integrity constraints.

8. For the following relational database, give the expressions in SQL:

branch schema (branch name, branch city, assets)
customer schema (customer name, cus- tomer street, customer city)
Loan schema (branch name, loan number, amount)
Borrower schema (customer name, Loan number)
Account schema (branch name, account number, balance)
Depositersecham (Customer name, account number)

a) Find the names of all customers whos street address include substring 'Main Building'
b) Find average balance for each customer who lives in Harrison and at least four accounts?
c) Find all customer who have a loan at bank whose names are neither 'smith' nor 'james'.

9. Consider the following schemas:

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

a) Find the name of sailors who have reserved boat 103.
b) Find the names and ages of sailors with a rating above 7.
c) Find the names of sailors who have reserved a red boat.
d) Find the sname, bid, and day for each reservation.
e) Find the name of sailors who have reserved at least one boat.

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Consider the following relation :**Cinema (theater, address, capacity)**
Which of the following options will be needed at the end of the SQL QUERY?

SELECT P1.address FROM Cinema P1, Such that it always finds the addresses of theaters of theaters with maximum capacity?

(a) WHERE P1.capacity > = All (select P2. capacity from Cinema P2)
(b) WHERE P1.capacity > = Any (select P2. capacity from Cinema P2)
(c) WHERE P1.capacity > All (select max (P2. capacity) from Cinema P2)
(d) WHERE P1.capacity >Any (select max (P2. capacity) from Cinema P2)

2. Consider the following relations:

Student	
Roll_No	Student_Name
1	Raj
2	Rohit
3	Raj

Performance		
Roll_No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

Consider the following SQL query:

SELECT S. Student_Name, sum (P.Marks)
FROM Student S, Performance P
WHERE S. Roll_No =P.Roll_No
GROUP BY S.Student_Name
The number of rows that will be returned by the SQL query is _____.

3. A relational schema for a train reservation database is given below:
Passenger (pid, pname, age)
Reservation (pid, class, tid)

Table: Passenger

pid	pname	age
0	Sachin	65
1	Rahul	66
2	Sourav	67
3	Anil	69

Table : Reservation

pid	class	tid
0	AC	8200
1	AC	8201
2	SC	8201
5	AC	8203
1	SC	8204
3	AC	8202

What pids are returned by the following SQL query for the above instance of the tables?

SELECT pid FROM Reservation WHERE class 'AC' ANDEXISTS (SELECT * FROM PassengerWHERE age > 65 ANDPassenger. pid = Reservation.pid)

1, 0 b)1, 2 c)1, 3 d)1, 5 [ ]

DATABASE MANAGEMENT SYSTEMS

Objectives:

To provide knowledge about the normalization techniques.

Learning Outcomes:

Students will be able to

- Understand various normalization techniques.
- Perform lossless decomposition and FD preserving on relations.
- To know how to overcome anomalies caused by redundancy.
- To know how to eliminate redundancy.
- Design the good database schema.

UNIT-III

Functional Dependencies: Partial, Full, Transitive and Trivial Dependencies, Axioms, Decomposition- Lossless join and Dependency Preserving Decomposition, Attribute closure, Normal forms- 1NF, 2NF, 3NF and BCNF.

Unit - III

Database Design

Functional Dependencies and Normalization for Relational Databases

1. Functional Dependencies

The single most important concept in relational schema design is a functional dependency.

A *Functional Dependency* describes a relationship between *attributes* within a single relation.

An attribute is *functionally dependent* on another if we can use the value of one attribute to determine the value of another. Formally, functional dependency is defined as;

Functional dependency – In agiven relation R, X and Y are attributes. Attribute Y is functionally dependent on attribute X if each value of X determines exactly one value of Y. This is represented as X → Y.

For example, consider the following relation

Reports(Student#, Course#, CourseName, Iname, Room#, Marks, Grade)

In this relation, {Student#, Course#} together can determine exactly one value of Marks. This can be symbolically represented as

{Student#, Course#} → Marks

This type of dependency is called as functional dependency. In the above example Marks is functionally dependent on {Student#, Course#}.

Other functional dependencies in the above example are:

- Course# →CourseName
- Course# →Iname (Assuming one course is taught by one and only one instructor)
- Iname→ Room# (Assuming each instructor has his/her own room)
- Marks → Grade
- Course# → Room#

Full Functional Dependency - In agiven relation R, X and Y are attributes. Attribute Y is fully functionally dependent on attribute X only if it is not functionally dependent on sub-set of X where X is composite in nature.

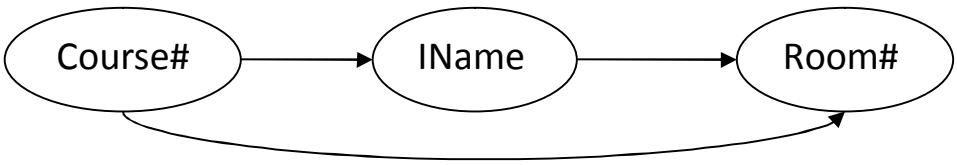
In the above example, Marks is fully functionally dependent on {Student#, Course#} and not on sub-set of {Student#, Course#}. This means Marks cannot be determined either by Student# or by Course#. It can be determined using Student# and Course# together. Hence, Marks is fully functionally dependent on {Student#, Course#}.

Partial Functional Dependency – In agiven relation R, X and Y are attributes. Attribute Y is partially dependent on attribute X only if it is dependent on sub-set of attribute X where X is composite in nature.

In the above example, CourseNmae, IName, Room# are partially dependent on {Student#, Course#} because Course# alone determines the CourseNmae, IName, Room#.

Transitive Dependency – In a given relation R, if attribute X determines attribute Y and attribute Y determines attribute Z, then attribute X determines attribute Z. Such a dependency is called transitive dependency.

Following example shows a transitive dependency.



Trivial Dependency-A FunctionalDependency X->Y is said to be trivial functional dependency if y is a subset of X(Y⊆X). In other words if R.H.S of some FD is the subset of L.H.S of FD is called Trivial Functional Dependency.

Example: AB->A

AB->B

AB->AB

2.Axioms: Armstrong's axioms are a set of rules, that when applied repeatedly generates a closure of functional dependencies.

1. Reflexive Rules: if X is a set of attributes and Y is a subset of X then X holds Y.

   $Y \subseteq X \Rightarrow X \rightarrow Y$

2. Augmentation Rule: if X hold Y and Z is a set of attributes then XZ holds YZ.

   X->Y then XZ->YZ

3. Transitive Rule: if X holds Y and Y holds Z , then X holds Z.

   X->Y and Y->Z then X->Z

4. Additive or Union Rule: if X holds Y and X holds Z ,then X holds YZ.

   X->Y and X->Z then X->YZ

5. Pseudo Transitive Rule: if X holds Y and YZ holds W, then XZ holds W.

   X->Y and YZ->W then XZ->W

6. Productive Rule or Decomposition Rule: if X holds YZ and X holds Y, then X holds Z.

   X->YZ then X->Y and X->Z

3.Decomposition:

- A functional decomposition is the process of breaking down the functions of an organization into progressively greater (finer and finer) levels of detail.
- In decomposition, one function is described in greater detail by a set of other supporting functions.
- The decomposition of a relation schema R consists of replacing the relation schema by two or more relation schemas that each contain a subset of the attributes of R and together include all attributes in R.
- Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistencies and anomalies.

There are two types of decomposition :(Properties of Decomposition)

1. **Lossless Join Decomposition**
2. **Dependency Preserving Decomposition**

Lossless Join Decomposition :

- "The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 yield the same relation as in R."
- A relational table is decomposed (or factored) into two or more smaller tables, in such a way that the designer can capture the precise content of the original table by joining the decomposed parts. This is called lossless-join (or non-additive join) decomposition.
- This is also referred as non-additive decomposition.
- The lossless-join decomposition is always defined with respect to a specific set F of dependencies.
- Consider that we have table STUDENT with three attribute roll_no ,sname and department.

  STUDENT :

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation Stu_name and Stu_dept :

Stu_name:

Roll_no	Sname
111	parimal
222	parimal

stu_dept

Roll_no	Dept
111	COMPUTER
222	ELECTRICAL

Now ,when these two relations are joined on the comman column 'roll_no' ,the resultant relation will look like stu_joined. stu_joined :

Roll_no	Sname	Dept
111	Parimal	COMPUTER
222	Parimal	ELECTRICAL

In lossless decomposition, no any spurious tuples are generated when a natural joined is applied to the relations in the decomposition.

Dependency Preservation Decomposition :

The dependency preservation decomposition is another property of decomposed relational database schema D in which each functional dependency X -> Y specified in F either appeared directly in one of the relation schemas R_i in the decomposed D or could be inferred from the dependencies that appear in some R_i.

Decomposition D = { R1 , R2, R3,,.., ,Rm} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each R_i , in D is equivalent to F. In other words, R join of R1, R1 over X. The dependencies are preserved because each dependency in F represents a constraint on the database. If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

Example:

Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.

A relation R is decomposed into -

R1 = (A, B, C) with FDs F1 = {A -> B, A -> C}, and

R2 = (C, D) with FDs F2 = {C -> D}.

F' = F1 ∪ F2 = {A -> B, A -> C, C -> D}

so, F' = F.

And so, F'+ = F+.

4.Attribute Closure:

The set of all those attributes which can be functionally determined from an attribute set is called as closure of that attribute set.

Closure of an attribute set {X} is denoted as $\{x\}^+$.

Steps to find closure of an attribute set:

1. Add the attributes contained in the attribute set for which closure is being calculated to the result set.
2. Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example: Consider a relation R(A,B,C,D,E,F,G) with the functional dependencies

A->BC

BC->DE

D->F

CF->G

Closure of Attribute A:

A+ ={A}

={ABC} ∵A->BC

={ABCDE} ∵BC->DE

={ABCDEF} ∵D->F

={ABCDEFG} ∵CF->G

Closure of Attribute set BC:

BC^+ ={BC}

={BCDE} ∵BC->DE

={BCDEF} ∵D->F

={BCDEFG} ∵CF->G

Normalization

Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion anomalies.

- These would include two properties
  - **Lossless join property**- which guarantees that the generation of spurious tuples will not occur.
  - **Dependency preservation property** - This ensures that each functional dependency is represented in some individual relation resulting after decomposition.
- **Prime attribute -** An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.
- **Non prime attribute -** An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.

If a relation schema has more than one key, each is called a candidate key. One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys.

Normal forms: The normal form is a relation refers to the highest normal form condition that it meets and hence indicates the degree to which it has been normalized.

Normal forms are used to eliminate or reduce redundancy in database tables.

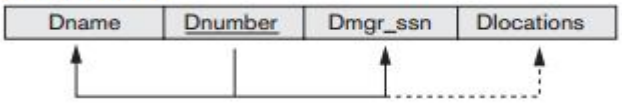
First Normal Form (1NF)

A relation R is said to be in the first normal form if and only if all the attributes of the relation R are atomic in nature.

For example, consider the Department table given in figure (b). It is not in 1NF because one of its attributes Dlocations is non-atomic as it contains more than one value in row 1. To make it 1NF compliant, create a separate row for each value of Dlocations of row 1 as shown in figure (c).

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure (a) A relation schema (b) sample state of relation Department that is not in 1NF (c)1NF version of the same relation

Second Normal Form (2NF)

A Relation is said to be in Second Normal Form (2NF) if and only if:

- It is in First normal form (1NF)
- No partial dependency exists between non-key attributes and key attributes

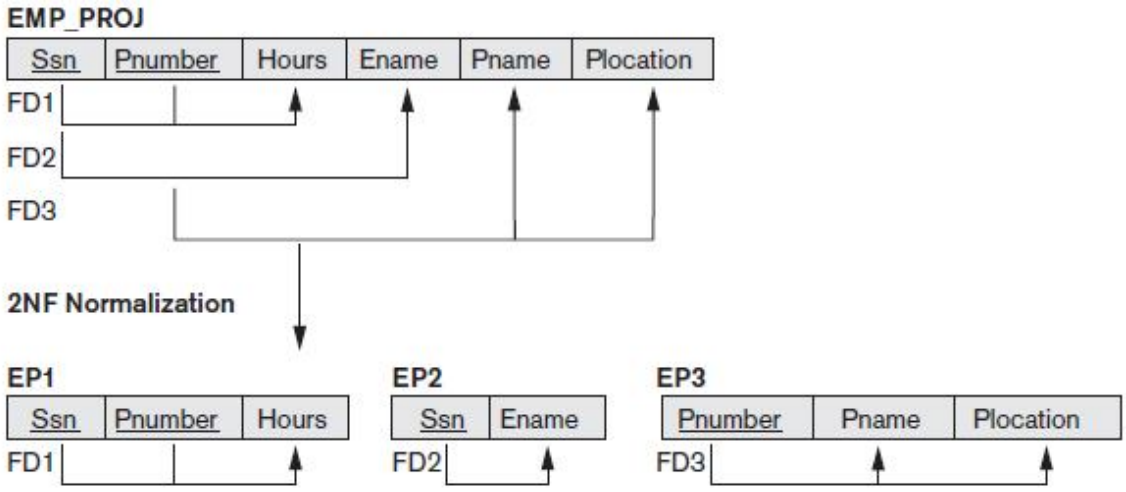
As an example, consider the EMP_PROJ schema shown below;

For this table, the key is {Ssn, Pnumber}

The functional dependencies are as follows;

- {Ssn, Pnumber} → Hours
- Ssn→Ename
- Pnumber→Pname
- Pnumber→Plocation

It is clear from these functional dependencies that the table has partial dependencies (Ssn→Ename, Pnumber→Pname, Pnumber→Plocation)



To make it 2NF compliant, remove all partial dependencies. For this, we need to split EMP_PROJ table into 3 tables as EP1, EP2 and EP3 as shown above.

Now these 3 tables do not contain partial dependencies and hence they are in 2NF.

Example2: Consider a relation R(A,B,C,D,E,F) with the functional dependencies

A->BCDEF

BC->ADEF

B->F

D->E

Solution:

Candidate keys are {A,BC}

So, prime attributes are {A,B,C}

Non-prime attributes are {D,E,F}

In the third functional dependency (B->F) partial dependency (non-prime attribute can't be derived from subset of candidate key) exists.

Therefore, the given relation R is not in 2NF

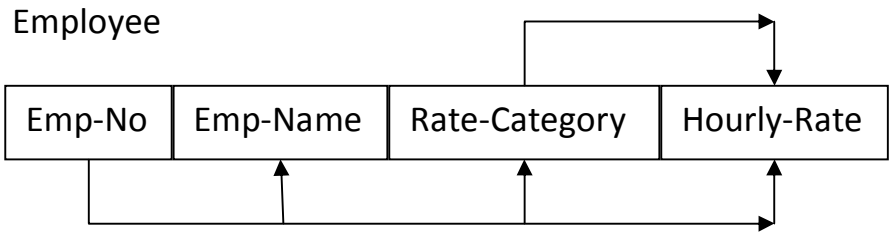
So, the relation R can be divided into two relations R1(A,B,C,D,E) and R2(B,F).

Third Normal Form (3NF):

A Relation is said to be in Third Normal Form (3NF) if and only if:

- It is in Second Normal Form (2NF)
- No transitive dependency exists between non-key attributes and key attributes

For example, consider the following Employee table.

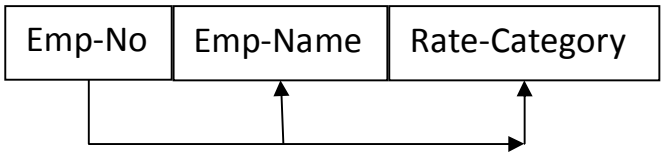


This table contains a transitive dependency as given below;

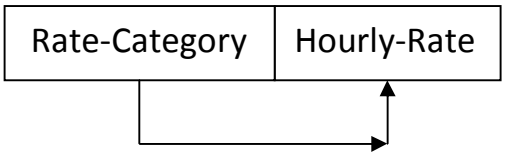
Emp-No → Rate-Category → Hourly-Rate

Hence, Employee table is not in 3NF. To make it 3NF compliant, we need to remove this transitive dependency. To do that, we need to split Employee table into two tables (Employee table and Rate table) as given below.

Employee Table



Rate Table



Now, both Employee and Rate tables are in 3NF as they do not have transitive dependencies.

Example2: Consider a relation R(A,B,C,D,E) with the functional dependencies

A->BCDE

BC->ADE

D->E

Solution:

Candidate keys are {A,BC}

So, prime attributes are {A,B,C}

Non-prime attributes are {D,E}

In the third functional dependency (D->E) Transitive dependency (non-prime attribute can't be derived from another non-prime attribute) exists.

Therefore, the given relation R is not in 3NF

So, the relation R can be divided into two relations R1(A,B,C,D) and R2(D,E).

Boyce-Codd Normal Form (BCNF):

A relation is said to be in BCNF if and only if all the determinants are candidate keys. BCNF relation is a strong 3NF relation. i.e. all BCNF relations are in 3NF but the reverse is not true.

For example, consider the following Result table.

Result Table

Student#	EmailID	Course#	Marks

This table has two candidate keys – {Student#, Course#} and {EmailID, Course#}.

This table is in 3NF because it has no partial and transitive dependencies between key attributes and non-key attributes. But this table is not in BCNF because all determinants are not candidate keys. This can be observed from the following FDs;

- {Student#, Course#} → Marks
- {EmailID, Course#} → Marks
- Student# →EmailID
- EmailID→ Student#

Though the determinants of first two FDs are candidate keys but the determinants of the last two FDs are not candidate keys. Thus it is violating the BCNF condition.

To make this table into BCNF compliant, we need to decompose the Result table into two tables as shown below;

Student Table

Student#	EmailID

Result Table

Student#	Course#	Marks

Example2: Consider a relation R(A,B,C,D) with the functional dependencies

A->BCD

BC->AD

D->B

Solution:

Candidate keys are {A,BC}

So, prime attributes are {A,B,C}

Non-prime attributes are {D}

The third functional dependency (D->B) Violates BCNF condition(because D is not a candidate key).

Therefore, the given relation R is not in BCNF

So, the relation R can be divided into two relations R1(A,C,D) and R2(B,D).

UNIT-III

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. The normalization of 1NF relations to 2NF involves: [ ]

   A)Removal of partial dependencies

   B) Removal of full dependencies

   C) Removal of transitive dependencies

   D) Removal of multi-valued dependencies

2. Why do we go for normalization of databases? [ ]

   A) To avoid the repetitions B) To prevent fragmentation

   C) To avoid redundancy D) To save memory

3. If a relation is in BCNF then it is in: [ ]

   A) 2 NF B) 3 NF C) 1 NF D) 1 NF and 2 NF

4. If a relation with a schema R is decomposed into two relations R1 and R2 such that (R1 ∪ R2) = R1 then which one of the following is to be satisfied for a lossless joint decomposition (→ indicates functional dependency) [ ]

   A) R1 ∩ R2 → R1 or R1 ∩ R2 → R2

   B) R1 ∩ R2 → R1

   C) R1 ∩ R2 → R2

   D) R1 ∩ R2 → R1 and R1 ∩ R2 → R2

5. Identify the minimal key for the relational scheme R(A, B, C, D, E) with functional dependencies F = {A→B, B→C, AC→D}. [ ]

   A) A B) AE C) BE D) CE

6. The best normal form of relation scheme R(A, B, C, D) along with the set of functional dependencies F = {AB→C, AB→D, C→A, D→B} is [ ]

   A) Boyce-Codd Normal form B) Third Normal form

   C) Second Normal form D) First Normal form

7. Match the following database terms to their functions: [ ]

List-I	List-II
(a) Normalization	(i) Enforces match of primary key to foreign key
(b) Data Dictionary	(ii) Reduces data redundancy in a database
(c) Referential Integrity	(iii) Define view(s) of the database for particular user(s).
(d) External Schema	(iv) Contains metadata describing database structure.

Codes:

	(a)	(b)	(c)	(d)
(A)	(iv)	(iii)	(i)	(ii)
(B)	(ii)	(iv)	(i)	(iii)
(C)	(ii)	(iv)	(iii)	(i)
(D)	(iv)	(iii)	(ii)	(i)

8. A relation R={A,B,C,D,E,F,G} is given with the following set of functional dependencies: F={AD→E, BE→F, B→C, AF→G}. [ ]
   Which of the following is a candidate key?
   A) A B) AB C) ABC D) ABD

9. Consider a relational schema R= (A, B, C, D, E, F, G, H) on which of the following functional dependencies hold: {A→B, BC→D, E→C, D→A}. [ ]

   What are the candidates keys for R?
   A) AE, BE B) AEH, BEH, DEH C) AEH, BEH, BCH D) AE, BE, DE

10. From the following instance of a relational schema R(A,B,C) we can conclude that

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

A) B does not functionally determines C

B) A does not functionally determine B and B does not functionally determines C

C) A functionally determine B and B functionally determines C

D) A functionally determine B and B does not functionally determines C

11. The relational schema student_performance(name, courseno, rollno, grade) has the following functional independencies. The highest normal form of this relation is_____.

{name, courseno}→grade

{rollno, courseno}→grade

name→rollno

rollno→name

12. Given the following relation instance [ ]

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the instance?

A) XY→Z and Y→X B) YZ→X and X→Z

C) XY→Z and Z→Y D) YZ→X and Y→Z

13. Relation R with an associated set of functional dependencies F, is decomposed into BCNF. The redundancy in the resulting set of relations is [ ]

A) Zero

B) more than zero but less than that of an equivalent 3 NF decomposition

C) proportional to the size of F

D) Indeterminate

SECTION-B

SUBJECTIVE QUESTIONS

1. Outline the informal guidelines for relational schema.
2. What are the problems caused by redundancy? Explain.

3. Define decomposition and illustrate lossless and dependency preserving decompositions.
4. What is normalization? Explain 1NF,2NF,3NF and BCNF.
5. Differentiate between 3NF and BCNF.
6. Define FD, MVD and JD.
7. Describe 4NF and 5NF with an example.
8. Consider the relation schema R(ABCD) and the FDs {AB->C, B->D}.What is the highest normal form condition it satisfies?
9. Consider the relation schema R(ABC) and the FDs {AB->C, C->A}. What is the highest normal form that it satisfies?
10. Consider the relation schema R(ABC) and the following FDs {AB->C, C->B}. What is the highest normal form condition it satisfies?
11. Given R(A,B,C,D, E) with the set of FDs, F{AB→CD, A→E, C→D}. Is the decomposition of R into R1(A, B, C), R2(B, C, D) and R3(C, D, E) lossless? Prove.
12. Given R(A,B,C,D,E) with the set of FDs, F{AB→CD, ABC→E, C→A}
    (i) Find any two candidate keys of R
    (ii) What is the normal form of R? Justify.
13. Let R = (A, B, C, D) and F be the set of functional dependencies for R given by {A→B, A→C, BC→D}. Prove A→D.
14. Given R = ABCD with the FD set F = { A→B, B→C, C→D}. Determine all 3NF violations. Decompose the relation into relations which are in 3NF.
15. Determine a candidate key for R = ABCDEG with the FD set
    F = {AB→C, AC→ B, AD→E, B→D, BC→A, E→G}

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. R(ABCD) is a relation. Which of the following does not have a lossless join, dependency preserving BCNF decomposition? **[GATE 2001]** **[ ]**

   A) A→B, B→C , C→D B) A→B, B→CD

   C) AB→C, C→AD D) A→BCD

2. The following functional dependencies are given below **[Gate 2005]**

   AB→CD, AF→D, DE→F, C→G, F→E, and G→A

   Which of the following option is false? [ ]

   A) {CF}+ = {ABCDEFG} B) {AF}+ = {ABCDEFG}

   C) {AB}+ = {ABCDFG} D) {BG}+ = {ABCDG}

3. Which of the following is TRUE? **[GATE 2012]**

   A) Every relation is 3NF is also in BCNF

   B) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R

   C) Every relation in BCNF is also in 3NF

   D) No relation can be in both BCNF and 3NF

4. Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values.

   F = {CH → G, A→ BC, B →CFH, E → A, F →EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R

   How many candidate keys does the relation R have? [**GATE 2013]** **[ ]**

   A) 3 B) 4 C) 5 D) 6

Unit - IV

Transaction Management

1. Transaction Concept

- **Transaction:** A **transaction** is a **unit** of program execution that accesses and possibly updates various data items. As an example consider the following transaction that transfers $50 from account A to account B. It can be written as:

  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

  This transaction consists of 6 actions that need to takes place in order to transfer $50 from account A to account B.

- **Comparing Transaction and Program:** A transaction is the result from the execution of user program written in high-level data manipulation language (DML) or programming language and it is started and ended between the statements **begin transaction** and **end transaction**.

- **Transaction operations**: Every transaction access data using two operations:
  - **read(X)** : which transfers the data item *X* from the database to a local buffer belonging to the transaction that executed the read operation.
  - **write(X)**: which transfers the data item *X* from the local buffer of the transaction that executed the write back to the database.

2. Properties of the Transaction

- To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:
  - **Atomicity**. Either all operations of the transaction are reflected properly in the database, or none are.
  - **Consistency**. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
  - **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.
    - That is, for every pair of transactions *Ti* and *Tj* , it appears to *Ti* that either *Tj* finished execution before *Ti* started, or *Tj* started execution after *Ti* finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
  - **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
- These properties are often called the **ACID properties**.
- To gain a better understanding of ACID properties and the need for them, consider a simplified banking system consisting of several accounts and a set of transactions that access and update those accounts.
- Let *Ti* be a transaction that transfers $50 from account *A* to account *B*. This transaction can be defined as;

T_i : read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

Let us now consider each of the ACID requirements.

- **Consistency:** The consistency requirement here is that the sum of A and B be unchanged by the execution of the transaction.
- It can be verified easily that, if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

- **Atomicity:** Suppose that, just before the execution of transaction Ti the values of accounts A and B are $1000 and $2000, respectively.
- Now suppose that, during the execution of transaction Ti, a failure occurs (power failures, hardware failures, and software errors) that prevent Ti from completing its execution successfully.
- Further, suppose that the failure happened after the write(A) operation but before the write(B) operation. In this case, the values of accounts A and B reflected in the database are $950 and $2000. The system destroyed $50 as a result of this failure. In particular, we note that the sum $A + B$ is no longer preserved.
- We call such a state as an **inconsistent state**. We must ensure that such inconsistencies are not visible in a database system.
- This state, however, is eventually replaced by the consistent state where the value of account A is $950, and the value of account B is $2050.

- This is the reason for the atomicity requirement: If the atomicity property is present, all actions of the transaction are reflected in the database, or none are.
- The basic idea behind ensuring atomicity is this: The database system keeps track (on disk) of the old values of any data on which a transaction performs a write, and, if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed.

- **Isolation:** If several transactions are executed concurrently, their operations may interleave in some undesirable way, resulting in an inconsistent state.
- For example consider the following; If between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

T1	**T2**
1. **read**(*A*)	
2. *A* := *A* – 50	
3. **write**(*A*)	
	read(A), read(B), print(A+B)
4. **read**(*B*)	
5. *B* := *B* + 50	
6. **write**(*B*)	

- A way to avoid the problem of concurrently executing transactions is to execute transactions **serially**—that is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, as we will see later.

- **Durability:** The durability property guarantees that, once a transaction completes successfully (i.e., the transfer of the $50 has taken place), all the updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution.

3. Transaction State

- **Active**, the initial state; the transaction stays in this state while it is executing
- **Partially committed**, after the final statement has been executed
- **Failed**, after the discovery that normal execution can no longer proceed
- **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction
- **Committed**, after successful completion

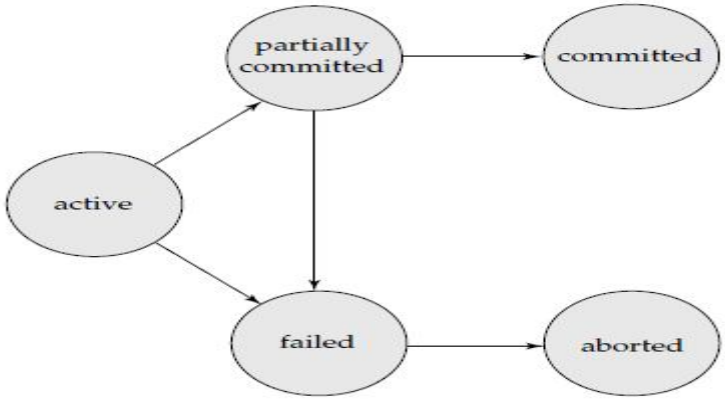


Figure State diagram of a transaction.

- ✓ In the absence of failures, all transactions complete successfully.
- ✓ A transaction may not always complete its execution successfully. Such a transaction is termed **aborted**. An **aborted** transaction must have no effect on the state of the database. Thus, any changes that the aborted transaction made to the database must be undone.
- ✓ Once the changes caused by an aborted transaction have been undone, we say that the transaction has been **rolled back**.

- ✓ A transaction that completes its execution successfully is said to be **committed**.
- ✓ A transaction enters the **failed state** after the system determines that the transaction can no longer proceed with its normal execution (for example, because of hardware or logical errors). Such a transaction must be **rolled back**. Then, it enters the **aborted state**. At this point, the system can either **restart** the transaction or **kill** the transaction.

4. Schedules

- A schedule is a sequences of instructions that specify the chronological order in which instructions of transactions are executed.
- **Serial Schedule:** If transactions are executed from start to finish, one after another then the schedule is called as a serial schedule.
- **Concurrent schedule:** If the instructions of different transactions are interleaved then the schedule is called as a concurrent schedule.
- Let *T*1 and *T*2 be two transactions that transfer funds from one account to another. Transaction *T*1 transfers $50 from account *A* to account *B*. It is defined as;

T1:
read(*A*);
A := *A* − 50;
write(*A*);
read(*B*);
B := *B* + 50;
write(*B*).

- Transaction *T*2 transfers 10 percent of the balance from account *A* to account *B*. It is defined as;

T2:
read(*A*);
temp := *A* * 0.1;
A := *A* − *temp*;

write(A);

read(B);

$B := B + temp$;

write(B).

- The following figure shows a serial schedule in which T1 executes before T2.

T_1	T_2
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)

Schedule 1—a serial schedule in which T_1 is followed by T_2.

- Similarly, the following figure shows a serial schedule in which T2 executes before T1.

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

Schedule 2—a serial schedule in which T_2 is followed by T_1.

- Thus, for a set of *n* transactions, there **exist *n*!** different valid serial schedules.
- If two transactions are running **concurrently**, the operating system may execute one transaction for a little while, then perform a context switch, execute the second transaction for some time, and then switch back to the first transaction for some time, and so on. With multiple transactions, the CPU time is shared among all the transactions.
- Concurrent execution of transactions improves the performance of the system. Following figure shows a concurrent schedule.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Schedule 3—a concurrent schedule equivalent to schedule 1.

- In case of concurrent execution, several execution sequences are possible, since various instructions from different transactions may now be interleaved. Thus, the number of possible schedules for a set of *n* transactions is much **larger than *n*!.**

- Not all concurrent executions result in a correct state. To illustrate, consider the schedule shown in following Figure. Suppose the current values of accounts A and B are \$1000 and \$2000, respectively. After the execution of this schedule, we arrive at a state where the final values of accounts A and B are \$950 and \$2100, respectively. This final state is an *inconsistent state*, since we have gained \$50 in the process of the concurrent execution and the sum $A + B$ is not preserved by the execution of the two transactions.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Schedule 4—a concurrent schedule.

- We can ensure consistency of the database under concurrent execution by making sure that any schedule that executed has the same effect as a schedule that could have occurred without any concurrent execution. The **concurrency-control component** of the database system carries out this task.
- A transaction that successfully completes its execution will have a commit instruction as the last statement.

- By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

5. Serializability

- The database system must control concurrent execution of transactions, to ensure that the database state remains consistent.
- For this, we must first understand which schedules will ensure consistency, and which schedules will not. For this, we need to consider simplified view of transactions.
- **Simplified view of transactions**
  - We ignore operations other than read and write instructions
  - We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
  - Our simplified schedules consist of only read and write instructions
- The following schedule consist only read and write instructions.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 3—showing only the read and write instructions.

- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
    1. Conflict serializability
    2. View serializability

5.1 Conflict Serializability

- Let us consider a schedule *S* in which there are two consecutive instructions *Ii* and *Ij*, of transactions *Ti* and *Tj* , respectively ($i \neq j$). If *Ii* and *Ij* refer to different data items, then we can swap *Ii* and *Ij* without affecting the results of any instruction in the schedule. If *Ii* and *Ij* refer to the same data item *Q*, then the order of the two steps may matter.
- Instructions *li* and *lj* of transactions *Ti* and *Tj* respectively, **conflict** if and only if there exists some item *Q* accessed by both *li* and *lj*, and at least one of these instructions wrote *Q*.
    1. *li* = **read**(*Q*), *lj* = **read**(*Q*). *li* and *lj* don't conflict.
    2. *li* = **read**(*Q*), *lj* = **write**(*Q*). They conflict.
    3. *li* = **write**(*Q*), *lj* = **read**(*Q*). They conflict
    4. *li* = **write**(*Q*), *lj* = **write**(*Q*). They conflict
- We say that *Ii* and *Ij* **conflict** if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.
- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of nonconflicting 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.
- To illustrate the concept of conflicting instructions, consider schedule 3, shown in the above figure. By performing a series of swaps of

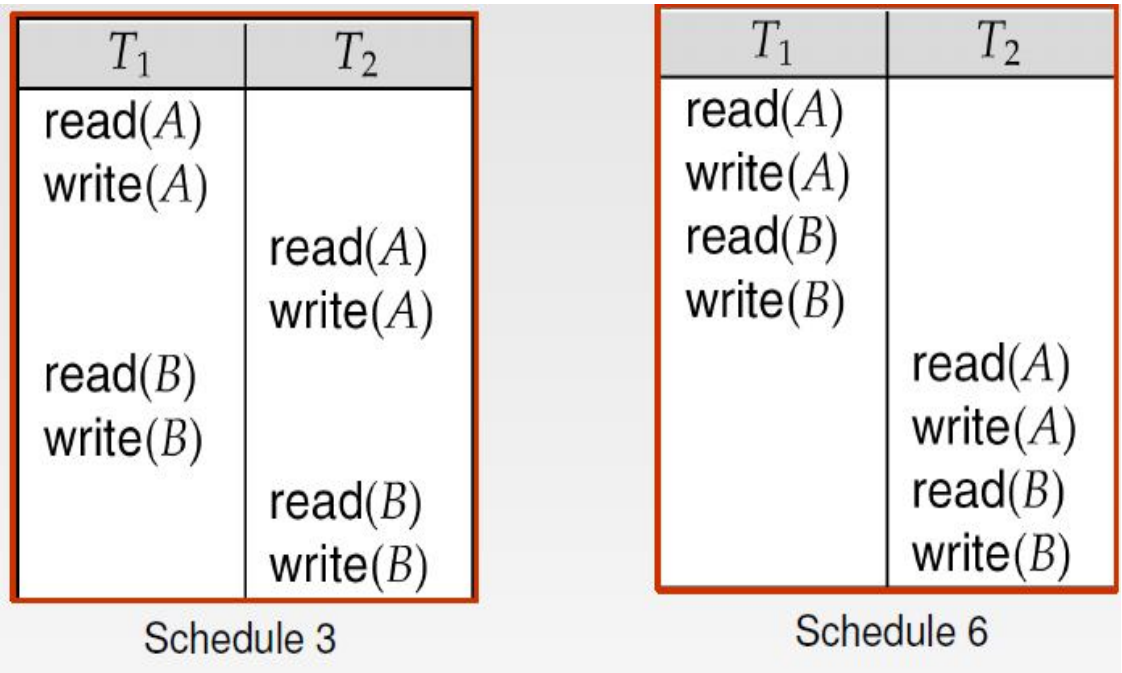
nonconflicting instructions, schedue 3 can be transformed to a serial schedule as follows;

- Swap read(B) of T1 with write(A) of T2. The result of this is shown beow;

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

- Similarly, by performing the following series of swaps of nonconflicting instructions, schedule 3 is transformed to a serial schedule as shown below;
    - Swap read(*B*) of *T*1 with read(*A*) of *T*2.
    - Swap write(*B*) of *T*1 with write(*A*) of *T*2.
    - Swap write(*B*) of *T*1 with read(*A*) of *T*2.



T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 3

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule 6

- The final result of these swaps, schedule 6 shown above, is a serial schedule. Thus, we have shown that schedule 3 is equivalent to a serial schedule and therefore schedule 3 is a conflict serializable schedule..
- Consider the schedule 7 of the following figure; it consists of only the read and write operations of transactions $T3$ and $T4$. This schedule is not conflict serializable, since it is not equivalent to either the serial schedule <$T3$,$T4$> or the serial schedule <$T4$,$T3$>.

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

Schedule 7

5.2 View Serializability

- Let *S* and *S*[1] be two schedules with the same set of transactions. The schedules *S* and *S*[1] are said to be **view equivalent** if three conditions are met, for each data item Q:
  1. If in schedule S, transaction *Ti* reads the initial value of *Q*, then in schedule *S'* also transaction *Ti* must read the initial value of *Q*.
  2. If in schedule S transaction *Ti* executes **read**(*Q*), and that value was produced by transaction *Tj* (if any), then in schedule *S'* also transaction *Ti* must read the value of *Q* that was produced by the same **write**(Q) operation of transaction *Tj* .
  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'*.

  As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is viewserializable but *not* conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Schedule 9 – A view serializable schedule

- Every view serializable schedule that is not conflict serializable has **blind writes**

Testing for Serializability

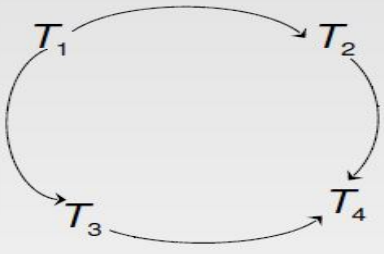
- Consider some schedule of a set of transactions *T*1, *T*2, ..., *Tn*
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from *Ti* to *Tj* if the two transaction conflict.
- A schedule is conflict serializable iff its precedence graph is acyclic.
- Examples;

T_1	T_2
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)

(a) (b)

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

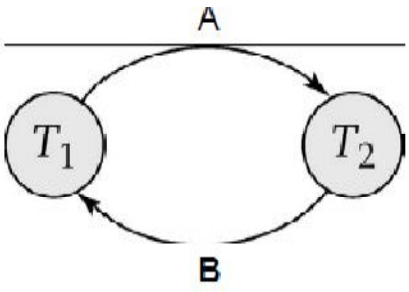
T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Schedule (c)

- The schedules shown above are conflict serializable schedules since, their precedence graphs are acyclic.
- The following schedule is not conflict serializable as its precedence graph consists a cycle.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)



The schedule cannot be conflict equivalent to a serial schedule

6. Recoverability

- If a transaction *Ti* fails, we need to undo the effect of this transaction to ensure the atomicity property of the transaction.
- In the following sections, we address the issue of what schedules are acceptable from the viewpoint of recovery from transaction failure.

6.1 Recoverable Schedules

- A **recoverable schedule** is one where, for each pair of transactions *Ti* and *Tj* such that *Tj* reads a data item previously written by *Ti*, the commit operation of *Ti* appears before the commit operation of *Tj* .

- The following schedule (schedule 11) is not a recoverable schedule if T9 commits immediately after the read(A) operation.

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

Schedule 11 – non-recoverable schedule

- If *T*8 should abort, *T*9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

6.2 Cascadeless Schedules

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks.
- Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable).

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Schedule 12

If *T*10 fails, *T*11 and *T*12 must also be rolled back.

- Cascading rollback lead to the undoing of a significant amount of work.
- A **cascadeless schedule** is one where, for each pair of transactions *Ti* and *Tj* such that *Tj* reads a data item previously written by *Ti*, the commit operation of *Ti* appears before the read operation of *Tj* .
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless.

UNIT-IV

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Identify the characteristics of transactions [ ]

A) Atomicity B) Durability C) Isolation D) All of the mentioned

2. Which of the following has "all-or-none" property? [ ]

A) Atomicity B)Durability C) Isolation D) All of the mentioned

3. Which one of the following is NOT a part of the ACID properties of database transactions? [ ]

A. Atomicity B) Isolation C) Consistency D) Deadlock-freedom

4. The database system must take special actions to ensure that transactions operate properly without interference from concurrently executing database statements. This property is referred to as: [ ]

A) Atomicity B) Durability C) Isolation D) All of the mentioned

5. The property of transaction that persists all the crashes is [ ]

A) Atomicity B)Durability C) Isolation D) All of the mentioned

6. Consider the following transaction involving two bank accounts x and y. read(x); x := x – 50; write(x); read(y); y := y + 50; write(y); The constraint that the sum of the accounts x and y should remain constant is known as:

A. Atomicity B) Isolation C) Consistency D) Durability[ ]

7. Precedence graphs help to find a [ ]

A) Serializable schedule C) Recoverable schedule

B) Deadlock free schedule D) Cascadeless schedule

8. Consider the following four schedules due to three transactions(indicated by the subscript) using read and write on a data item x, denoted by r(x) and w(x) respectively. Which one of them is conflict serializable? [ ]

(A) $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_3(x)$; $w_2(x)$

(B) $r_2(x)$; $r_1(x)$; $w_2(x)$; $r_3(x)$; $w_1(x)$

(C) $r_3(x)$; $r_2(x)$; $r_1(x)$; $w_2(x)$; $w_1(x)$

(D) $r_2(x)$; $w_2(x)$; $r_3(x)$; $r_1(x)$; $w_1(x)$

9. Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below. [ ]

T1: r1(X); r1(Z); w1(X); w1(Z)

T2: r2(Y); r2(Z); w2(Z)

T3: r3(Y); r3(X); w3(Y)

S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z); w3(Y); w2(Z); r1(Z); w1(X); w1(Z)

S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z); r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

Which one of the following statements about the schedules is TRUE?

A. Only S1 is conflict-serializable.

B. Only S2 is conflict-serializable.

C. Both S1 and S2 are conflict-serializable.

D. Neither S1 nor S2 is conflict-serializable.

10. Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects {O1,...,Ok}. This is done in the following manner:

Step 1. T acquires exclusive locks to O1, . . . , Ok in increasing order of their addresses. Step 2. The required operations are performed. Step 3. All locks are released. This protocol will; [ ]

A. guarantee serializability and deadlock-freedom

B. guarantee neither serializability nor deadlock-freedom

C. guarantee serializability but not deadlock-freedom

D. guarantee deadlock-freedom but not serializability

11. Which of the following property state that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. [ ]

A) Consistency B) Atomicity C) Durability D) Isolation

12. Which property states that only valid data will be written to the database?

A. Consistency B) Durability C) Atomicity D) Isolation

SECTION-B

SUBJECTIVE QUESTIONS

1) Define Transaction and briefly explain ACID properties.
2) Draw transaction state diagram and describe each state that a transaction goes through during its execution.
3) What is schedule? Explain different types of schedules.
4) How can you test whether a given schedule is conflict-serializable? Is every conflict-serializable schedule is serializable? Justify.
5) Construct a precedence graph for serial schedule and non serial schedule.
6) Create a concurrent schedule for executing the following transactions;

   T1: transfer funds $1000 from account A to account B

   T2: Increase the balance amount of account A to 10%
7) Consider the following two transactions:

   *T*1: read(*A*);

   read(*B*);

   if *A* = 0**then** *B* := *B* + 1;

   write(*B*).

   *T*2: read(*B*);

   read(*A*);

   if *B* = 0**then** *A* := *A* + 1;

   write(*A*).

   Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.
b. Show a concurrent execution of *T*1 and *T*2 (shown in the above problem) that produces a nonserializable schedule.

8) Is there a concurrent execution of *T*1 and *T*2 (shown in the above problem) that produces a serializable schedule?

9) Test whether the following schedule is conflict serializable (Subscripts denote transactions)?

S1: R1(X); R2(X); W1(X); R3(X); W2(X);

10) Consider the following three schedules due to three transactions (indicated by the subscript) using read and write on a data item X, denoted by R(X) and W(X) respectively. Construct the precedence graph for each schedule and determine which of them is conflict serializable?

S1: R2(X); R1(X); W1(X); R3(X); W2(X);

S2: R3(X); R2(X); R1(X); W2(X); W1(X);

S3: R2(X); W2(X); R3(X); R1(X); W1(X);

11) Consider three transactions: T1, T2 and T3. Draw the precedence graph for the following schedule consisting of these three transactions and determine whether it is serializable. If so, give its serial order(s).

T1	T2	T3
		read(Y)
		read(Z)
read(X)		
write(X)		
		write(Y)
		write(Z)
	read(Z)	
read(Y)		
write(Y)		

	read(Y) write(Y) read(X) write(X)	

SECTION-C
QUESTIONS AT THE LEVEL OF GATE

1. Consider the following schedules involving two transactions. Which one of the following statements is TRUE? (GATE 2007)

   S_1: $r_1(X)$; $r_1(Y)$; $r_2(X)$; $r_2(Y)$; $w_2(Y)$; $w_1(X)$

   S_2: $r_1(X)$; $r_2(X)$; $r_2(Y)$; $w_2(Y)$; $r_1(Y)$; $w_1(X)$

   A) Both S_1 and S_2 are conflict serializable

   B) S_1 is conflict serializable and S_2 is not conflict serializable

   C) S_1 is not conflict serializable and S_2 is conflict serializable

   D) Both S_1 and S_2 are not conflict serializable

2. Consider the following schedule **S** of transactions T1, T2, T3, T4:

T1	T2	T3	T4
	Reads(X)		
		Writes(X)	
		Commit	
Writes(X)			
Commit			
	Writes(Y)		
	Reads(Z)		
	Commit		

			Reads(X) Reads(Y) Commit

Which one of the following statements is CORRECT? (GATE 2014)

A) **S** is conflict-serializable but not recoverable

B) **S** is not conflict-serializable but is recoverable

C) **S** is both conflict-serializable and recoverable

D) **S** is neither conflict-serializable nor is it recoverable

3. Consider the following transactions with data items P and Q initialized to zero: (GATE 2012)

T1: read (P) ;
read (Q) ;
if P = 0 then Q : = Q + 1 ;
write (Q).

T2: read (Q) ;
read (P)
if Q = 0 then P : = P + 1 ;
write (P).

Any non-serial interleaving of T1 and T2 for concurrent execution leads to

A) a serializable schedule

B) a schedule that is not conflict serializable

C) a conflict serializable schedule

D) a schedule for which precedence graph cannot be drawn

4. Which of the following scenarios may lead to an irrecoverable error in a database system? (GATE 2003)
   A) A transaction writes a data item after it is read by an uncommitted transaction
   B) A transaction reads a data item after it is read by an uncommitted transaction
   C) A transaction reads a data item after it is written by a committed transaction
   D) A transaction reads a data item after it is written by an uncommitted transaction

4. Consider the data items D1, D2 and D3, and the following execution schedule of transactions T1, T2, and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively.

T1	T2	T3
	R(D3)	
	R(D2)	
	W(D2)	
		R(D2)
		R(D3)
R(D1)		
W(D1)		
		W(D2)
		W(D3)
	R(D1)	
R(D2)		

W(D2)	W(D1)	

Which of the following statements is correct?

A) The schedule is searializable as T2, T3, T1

B) The schedule is searializable as T2, T1, T3

C) The schedule is searializable as T3, T2, T1

D) The schedule is not searializable

****

Unit - V

Concurrency Control

1. Concurrent Execution of Transactions

- Transaction-processing systems usually allow multiple transactions to run concurrently. Allowing multiple transactions to update data concurrently causes several complications. However, there are two good reasons for allowing concurrency:
  - **Improved throughput and resource utilization**. A transaction consists of many steps. Some involve I/O activity; others involve CPU activity. The CPU I/O activities can be done in parallel. While a read or write on behalf of one transaction is in progress on one disk, another transaction can be running in the CPU, while another disk may be executing a read or write on behalf of a third transaction. All of this increases the **throughput** of the system—that is, the number of transactions executed in a given amount of time. Correspondingly, the processor and disk **utilization** also increase.
  - **Reduced waiting time**. There may be a mix of transactions running on a system, some short and some long. If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction. Concurrent execution reduces the unpredictable delays in running transactions. Moreover, it also reduces the **average response time**.

2. Anomalies Due to Concurrent Executions

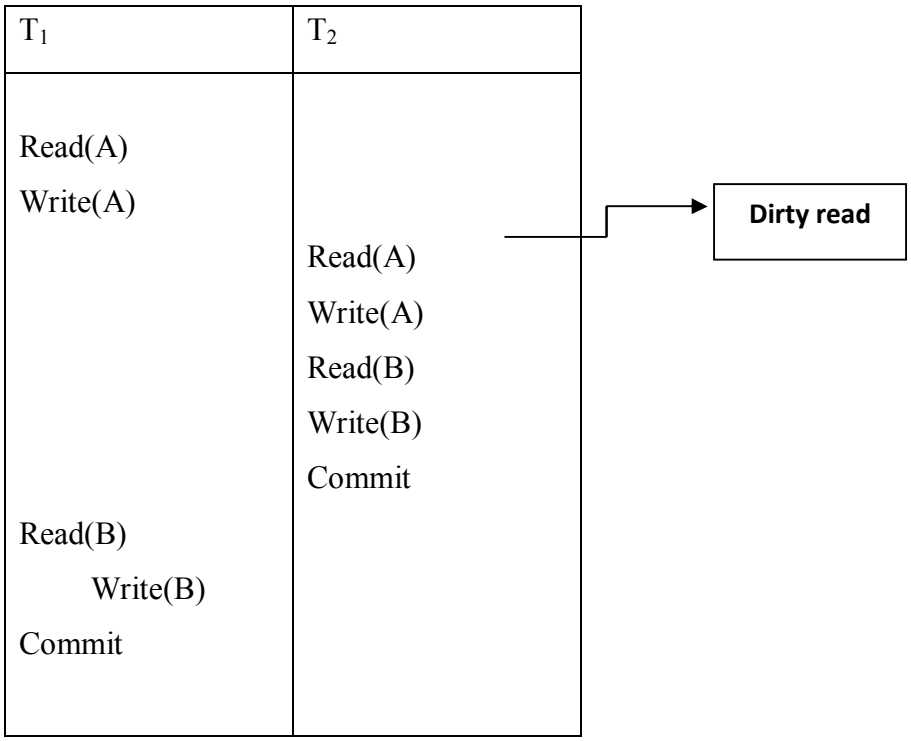
- If all the transactions in DBMS systems are doing read operation on the Database then no problem will arise. When the read and write operations are done simultaneously, then there is a possibility of some type of anomalies. These are classified into three categories.
  - Write–Read Conflicts (WR Conflicts)
  - Read–Write Conflicts (RW Conflicts)
  - Write–Write Conflicts (WW Conflicts)

WR Conflicts (or Dirty Read)

- This happens when the transaction T_2 read the data item A that has been modified by another transaction T_1, which has not yet committed. Such a read is called a *dirty read*.
- As an example, consider the following set of transactions (T_1 & T_2) and the schedule for executing T_1 & T_2.

  T_1: Transfers $100 from account A to account B

  T_2: Increments the balances of account A and account B by 10%



Reading uncommitted data

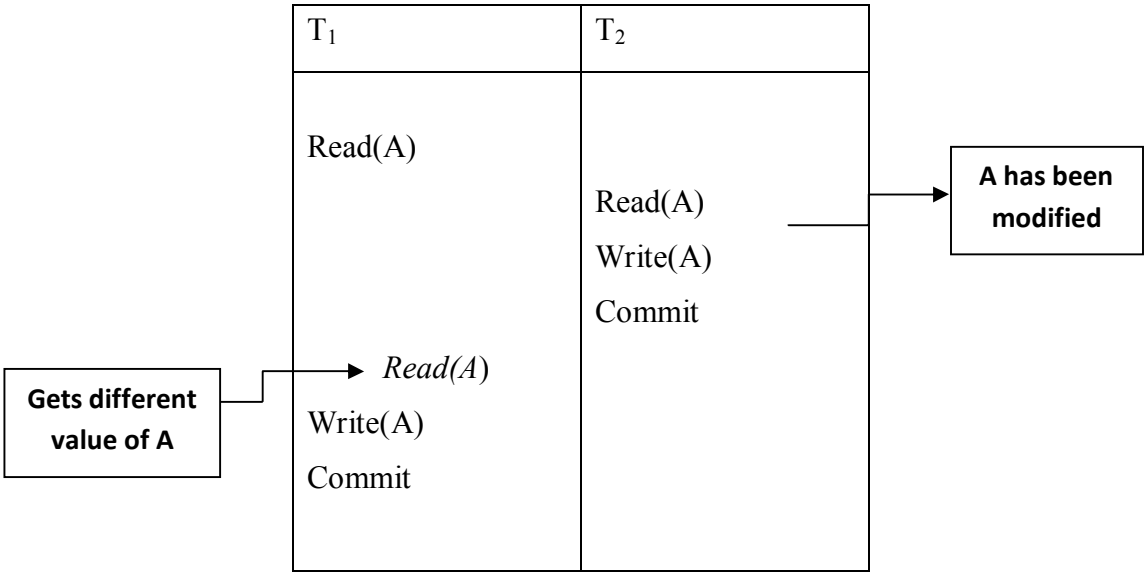
If the two transactions are allowed to execute as per the schedule shown above, then the outcome of this execution will be different from the normal execution like if the two instructions are executed one after another. This type of anomalies leaves the database in an inconsistency state.

RW Conflicts (or Unrepeatable Read)

- This happens when the transaction T_2 has modified the data item A that has been read by another transaction T_1, while T_1 is still in progress. If T_1 tries to read the value of A again, it will get a different value, even though it has not modified A in the mean time. This is called *unrepeatable read*.
- As an example, consider the following set of transactions (T_1 & T_2) and the schedule for executing T_1 & T_2.

  T_1: Reduce account A by $100

  T_2: Reduce account B by $100



RW conflict

Again, the execution of transactions T_1 & T_2 using the above schedule leads to database inconsistency.

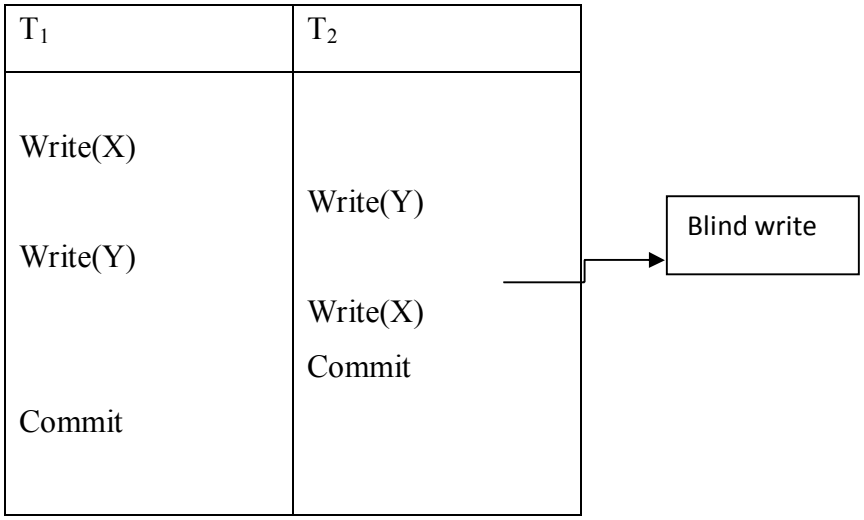
WW Conflicts (or Blind Write)

- This happens when the transaction T_2 could overwrite the value of data item A, which has already been modified by another transaction T_1, while T_1 is still in progress. Such a write is called *blind write*.

- As an example, consider the following set of transactions (T_1 & T_2) and the schedule for executing T_1 & T_2.

  T_1: Sets the salaries of X and Y to $1000

  T_2: Sets the salaries of X and Y to $2000



WW conflict

Again, the execution of transactions T_1 & T_2 using the above schedule leads to database inconsistency.

3. Lock-Based Protocols

- One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item no other transaction can modify that data item. A DBMS typically uses a *locking protocol* to achieve this.

Locks

- There are various modes in which a *data item may be locked*.
  1. **Shared Lock:** If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write, Q.
  2. **Exclusive Lock:** If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

Following figure shows compatibility of locks.

	S	X
S	true	false
X	false	false

- From the figure it is clear that shared lock is compatible with shared lock, but not with exclusive lock. At any time, several shared-mode locks can be held simultaneously (by different transactions) on a particular data item. A subsequent exclusive-mode lock request has to wait until the currently held shared-mode locks are released.

The Two-Phase Locking Protocol (2PL)

- It ensures serializability.
- This protocol requires that each transaction issue lock and unlock requests in two phases:
  1. **Growing phase.** A transaction may obtain locks, but may not release any lock.
  2. **Shrinking phase.** A transaction may release locks, but may not obtain any new locks.
- Initially, a transaction is in the *growing phase*. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the *shrinking phase*, and it can issue no more lock requests.

T_1	T_2
Lock-X(A)	
Read(A)	
Write(A)	
Lock-X(B)	
Unlock(A)	
	Lock-X(A)
	Read(A)
	Write(A)
Read(B)	
Write(B)	
Unlock(B)	

	Lock-X(B) Unlock(A) Read(B) Write(B) Unlock(B)

Schedule – 1

- As an example, consider the following set of transactions (T_1 & T_2)

  T_1: Transfers $100 from account A to account B

  T_2: Increments the balances of account A and account B by 10%

  T_1 & T_2 can be executed using **schedule – 1** under Two-phase locking protocol.

Advantages

- It ensures serializability
- It is a simple and straightforward protocol.

Disadvantages

- Deadlocks may occur.

  Consider the following partial schedule. This schedule is under Two-phase locking protocol. However it leads to a deadlock since *T*3 is holding an exclusive-mode lock on *B* and *T*4 is requesting a shared-mode lock on *B*, *T*4 is waiting for *T*3 to unlock *B*. Similarly, since *T*4 is holding a shared-mode lock on *A* and *T*3 is requesting an exclusive-mode lock on *A*, *T*3 is waiting for *T*4 to unlock *A*. Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This situation is called **deadlock**.

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Cascading rollbacks may occur.

  Consider the following partial schedule. Each transaction in this schedule is observing Two-phase locking protocol. However it leads to cascading rollbacks. If T_5 aborts after *read(A)* of T_7 then it leads to the rollback of T_5 which in turn leads to the rollback of T_6 and T_7 as T_6 has read the modified value of A by T_5 and T_7 has read the modified value of A by T_6. If rollback of a transaction causes rollback of a series of transactions then such rollbacks are called as cascading rollbacks.

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

Strict two-phase locking protocol (Strict 2PL)

- Cascading rollbacks can be avoided by a modification of two-phase locking called the **strict two-phase locking protocol**.

- It also ensures serializability.
- It has the following rules;

  Rule 1: It is similar to two-phase locking protocol.

  Rule 2: All exclusive locks held by a transaction are released only when the transaction commit/abort.

- As an example, consider the following two transactions;

  T_1: Transfers $100 from account A to account B

  T_2: Increments the balances of account A and account B by 10%

  The following schedule results for executing T_1 & T_2 under Strict Two-phase locking protocol.

T_1	T_2
Lock-X(A)	
Read(A)	
Write(A)	
Lock-X(B)	
Read(B)	
Write(B)	
commit	
Unlock(A)	
Unlock(B)	
	Lock-X(A)
	Read(A)
	Write(A)
	Lock-X(B)
	Read(B)
	Write(B)
	commit
	Unlock(A)
	Unlock(B)

Rigorous two-phase locking protocol

- Another variant of two-phase locking is the **rigorous two-phase locking protocol.**
- It also ensures serializability.
- It has the following rules;

  Rule 1: It is similar to two-phase locking protocol.
  Rule 2: All locks held by a transaction are released only when the transaction commit/abort.
- With rigorous two-phase locking, transactions can be serialized in the order in which they commit. Most database systems implement either strict or rigorous two-phase locking.
- As an example, consider the following two transactions;
  T_1: Transfers $100 from account A to account B
  T_2: Increments the balances of account A and account B by 10%
  The following schedule results for executing T_1 & T_2 under Strict Two-phase locking protocol.

T_1	T_2
Lock-X(A)	
Read(A)	
Write(A)	
Lock-X(B)	
Read(B)	
Write(B)	
commit	
Unlock(A)	
Unlock(B)	
	Lock-X(A)
	Read(A)
	Write(A)
	Lock-X(B)
	Read(B)
	Write(B)
	commit
	Unlock(A)

	Unlock(B)

4. Timestamp-Based Protocols

- Another method for determining the serializability order is a **timestamp-ordering** scheme.

Timestamps

- With each transaction Ti in the system, we associate a unique fixed timestamp, denoted by TS(Ti). This timestamp is assigned by the database system before the transaction Ti starts execution.
- Usually, the value of the system clock is used as the timestamp; that is, a transaction's timestamp is equal to the value of the system clock when the transaction enters the system.
- The timestamps of the transactions determine the serializability order. Thus, if TS(Ti) < TS(Tj), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction Tj .
- To implement this scheme, we associate with each data item Q two timestamp values:
    a) **W-timestamp(Q)** denotes the largest timestamp of any transaction that executed write(Q) successfully
    b) **R-timestamp(Q)** denotes the largest timestamp of any transaction that executed read(Q) successfully.
- These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

The Timestamp-Ordering Protocol

- The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:
    1. Suppose that transaction *Ti* issues read(*Q*).
        a. If TS(*Ti*) < W-timestamp(*Q*), then *Ti* needs to read a value of *Q* that was already overwritten. Hence, the read operation is rejected, and *Ti* is rolled back.
        b. If TS(*Ti*) ≥ W-timestamp(*Q*), then the read operation is executed, and R-timestamp(*Q*) is set to the maximum of R-timestamp(*Q*) and TS(*Ti*).

2. Suppose that transaction *Ti* issues write(*Q*).
    a. If TS(*Ti*) < R-timestamp(*Q*), then the value of *Q* that *Ti* is producing was needed previously. Hence, the system rejects the write operation and rolls *Ti* back.
    b. If TS(*Ti*) < W-timestamp(*Q*), then *Ti* is attempting to write an obsolete value of *Q*. Hence, the system rejects this write operation and rolls *Ti* back.
    c. Otherwise, the system executes the write operation and sets W-timestamp(*Q*) to TS(*Ti*).

- If a transaction *Ti* is rolled back by the concurrency-control scheme as a result of issuance of either a read or write operation, then the system assigns it a new timestamp and restarts it.
- To illustrate this protocol, we consider the following two transactions.

  T_{14}: Displays the contents of accounts A and B

  T_{15}: Transfers $50 from account A to account B, and then displays the contents of both.
- Following schedule results for executing T_{14} and T_{15} under Timestamp-based protocol.

T_{14}	T_{15}
read(B)	
	read(B)
	$B := B - 50$
	write(B)
read(A)	
	read(A)
display($A + B$)	
	$A := A + 50$
	write(A)
	display($A + B$)

- The main advantage of Timestamp-based protocol is that it is free from deadlocks.

Thomas' Write Rule

- Consider the following schedule and apply the timestamp-ordering protocol.

T_{16}	T_{17}
read(Q)	
	write(Q)
write(Q)	

- Since T_{16} starts before T_{17}, we shall assume that TS(T_{16}) < TS(T_{17}). The read(Q) operation of T_{16} succeeds, as does the write(Q) operation of T_{17}.When T_{16} attempts its write(Q) operation, we find that TS(T_{16}) < W-timestamp(Q). Thus, the write(Q) by T_{16} is rejected and transaction T_{16} must be rolled back.
- Although the rollback of T_{16} is required by the timestamp-ordering protocol, it is unnecessary. Since T_{17} has already written Q, the value that T_{16} is attempting to write is one that will never need to be read. Any transaction Ti with TS(Ti) < TS(T_{17}) that attempts a read(Q) will be rolled back, since TS(Ti) < W-timestamp(Q). Any transaction Tj with TS(Tj) > TS(T_{17}) must read the value of Q written by T_{17}, rather than the value written by T_{16}.
- This observation leads to a modified version of the timestamp-ordering protocol in which obsolete write operations can be ignored under certain circumstances.
- The modification to the timestamp-ordering protocol is called Thomas' write rule and is given as;
- Suppose that transaction Ti issues write(Q)
    1. If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was previously needed. Hence, the system rejects the write operation and rolls Ti back.
    2. If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation can be ignored.
    3. Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(Ti).
- The difference between these rules and the timestamp-ordering protocol lies in the second rule. The timestamp-ordering protocol requires that *Ti* be rolled back if *Ti* issues write(Q) and TS(Ti) < W-timestamp(Q). However, here, in Thomas' Write Rule, we ignore the obsolete write.

- Thomas' write rule makes use of view serializability and increases concurrency.

6. Deadlock Handling

- A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- For example there exists a set of waiting transactions {T0, T1, . . ., Tn} such that T0 is waiting for a data item that T1 holds, and T1 is waiting for a data item that T2 holds, and . . ., and Tn−1 is waiting for a data item that Tn holds, and Tn is waiting for a data item that T0 holds. None of the transactions can make progress in such a situation.
- There are two principal methods for dealing with the deadlock problem.
    1. deadlock prevention
    2. deadlock detection and deadlock recovery scheme.

Deadlock Prevention

- Two different deadlock prevention schemes using timestamps have been proposed:

a. The **wait–die** scheme is a non preemptive technique. When transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp smaller than that of Tj (that is, Ti is older than Tj). Otherwise, Ti is rolled back (dies).

For example, suppose that transactions T22, T23, and T24 have timestamps 5, 10, and 15, respectively. If T22 requests a data item held by T23, then T22 will wait. If T24 requests a data item held by T23, then T24 will be rolled back.

b. The **wound–wait** scheme is a preemptive technique. It is a counterpart to the wait–die scheme. When transaction Ti requests a data item currently held by Tj , Ti is allowed to wait only if it has a timestamp larger than that of Tj (that is, Ti is younger than Tj). Otherwise, Tj is rolled back (Tj is wounded by Ti).

Example, with transactions T22, T23, and T24, if T22 requests a data item held by T23, then the data item will be preempted from T23, and T23 will be rolled back. If T24 requests a data item held by T23, then T24 will wait.

- The major problem with both of these schemes is that *unnecessary rollbacks may occur.*

Deadlock Detection and Recovery

- An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred. If one has, then the system must attempt to recover from the deadlock. This uses a directed graph called a **wait-for graph** for Deadlock Detection.
- This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges.
- The set of **vertices** represents the **transactions** in the system and there is an **edge form T_i to T_j if T_i waits for T_j**.
- A **deadlock exists** in the system if and only if the **wait-for graph contains a cycle**.

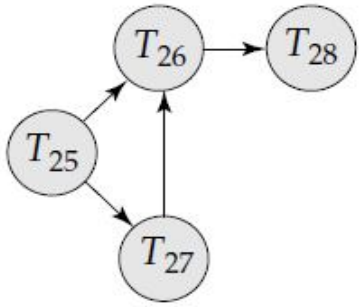


Fig. 3: Waits-for graph

- To illustrate these concepts, consider the wait-for graph in Figure 3, which depicts the following situation:
  - Transaction T25 is waiting for transactions T26 and T27.
  - Transaction T27 is waiting for transaction T26.
  - Transaction T26 is waiting for transaction T28.
- Since the graph has no cycle, the system is not in a deadlock state.

- Suppose now that transaction T28 is requesting an item held by T27. The edge T28 → T27 is added to the wait-for graph, resulting in the new system state in Figure 4. This time, the graph contains the cycle

  T26 → T28 →T27 →T26 implying that transactions T26, T27, and T28 are all deadlocked.

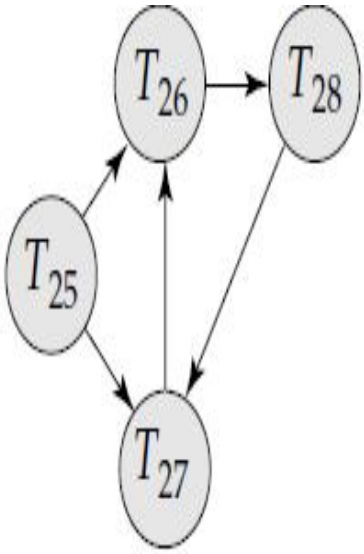


Fig. 4: Waits-for graph

Recovery from Deadlock

- When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution for deadlock recovery is to roll back one or more transactions to break the deadlock. Three actions need to be taken:
  1. **Selection of a victim**: Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Factors that determine the cost of a rollback, including
     a. How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.
     b. How many data items that the transaction has used.
     c. How many more data items the transaction needs for it to complete.

d. How many transactions will be involved in the rollback.

2. **Rollback:** Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back.
    a. **Total rollback**: Abort the transaction and then restart it.
    b. **Partial rollback:** roll back the transaction only as far as necessary to break the deadlock.
3. **Starvation:** In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is **starvation**. We must ensure that transaction can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

UNIT-V

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. If a transaction acquires a shared lock, then it can perform_____operation. [ ]
   A) Read B) Write C) Read and Write D) Update
2. If a transaction obtains an exclusive lock on a row, it means that the transaction wants to _____that row. [ ]
   A) Select B) Update C) View D) Read
3. In a two-phase locking protocol, a transaction release locks in _____phase. [ ]

A) Shrinking phase B) Growing phase C) Running phase D) Initial phase

4. In time stamp based protocol, transactions are executed based on their_____
5. _____protocol ensure that the system will never enter into a deadlock state.
6. Deadlocks can be described precisely in terms of a directed graph called _____
7. In strict 2PL [ ]

1. Locking be in 2PL
2. All exclusive locks must be held until transaction commits
3. All shared and exclusive locks must be held until transaction commits

A) Both 1 and 2 B) Only 2 C) Only 2 D) All of the above

8. In rigorous 2PL [ ]

1. Locking be in 2pl
2. All shared and exclusive locks must be held until transaction commits

A) Both 1 and 2 B) Only 2 C) Only 1 D) None of the above

9. Two phase locking doesn't ensure [ ]

A) Freedom from deadlock B) Cascading rollbacks

C) Both a and b D) Either a or b

10. Test whether the following schedule observes i) 2PL ii) Strict 2pl iii) Rigorous 2PL [ ]

Lock- S(A)

R(A)

Lock –X(B)

R(B)

Unlock(A)

W(B)

Unlock(B)

A) Only I B) I & II C) I, II, & III D) None

11. Test whether the following schedule observes i) 2PL ii) Strict 2pl iii) Rigorous 2PL [ ]

Lock- S(A)

R(A)

Lock –X(B)

Unlock (A)

R(B)

W(B)

Commit

Unlock (B)

A) Only I B) I & II C) I, II, & III D) None

12. Test whether the following schedules observes i) 2PL ii) Strict 2pl iii) Rigorous 2PL [ ]

Lock- S(A)

R(A)

Lock –X(B)

Write(B)

Unlock(A)

Unlock(B)

A) Only I B) I & II C) I, II, & III D) None

SECTION-B

Descriptive Questions

1. Why concurrency control is needed? Explain the problems that would arise when concurrency control is not provided by the database system.
2. Identify the anomalies due to concurrent execution of transactions. (Dirty Read, Unrepeatable Read, Blind Write)
3. What is a lock? List the types of lock.
4. Define 2-Phase Locking. Differentiate 2PL, Strict 2PL and Rigorous 2PL.
5. Discuss in detail about Time Stamp Based Protocol and Thomas Write Rule.
6. What is deadlock? Illustrate different deadlock handling techniques.
7. Outline the actions to be taken to recover from a deadlock.
8. Draw the waits-for graph for the following schedule and test whether this schedule leads to a deadlock?

T1	T2	T3	T4
Lock-S(A)			
Read(A)			
	Lock-X(B) Write(B)		
Lock-S(B)		Lock-S(C)	
	Lock-X(C)	Read(C)	
			Lock-X(B)
		Lock-X(A)	

9. Describe wait/die and wound/wait deadlock protocols.
10. Consider the following schedules;

    S1: T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)

    S2: T1:R(X),T1:R(Y),T1:W(X),T2:R(Y),T3:W(Y),T1:W(X),T2:R(Y)

For each of the above schedules, state whether timestamp-based protocol allows the actions to occur in exactly the order shown.

SECTION-C

GATE Questions

1) Which of the following concurrency control protocols ensure both conflict serialzability and freedom from deadlock? [GATE 2010]

I) 2-phase locking

II) Time-stamp ordering [ ]

A) I only B) II only C) Both I and II D) Neither I nor II

UNIT VI

Crash Recovery & Indexing

1. Failure Classification

There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. Broadly failures are classified into the following types.

- **Transaction failure.** There are two types of errors that may cause a transaction to fail:
  1. **Logical error.** The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.
  2. **System error.** The system has entered an undesirable state (deadlock), so the transaction cannot continue with its normal execution. The transaction, however, can be re executed at a later time.
- **System crash.** There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of nonvolatile storage remains intact, and is not corrupted.
  - The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the **fail-stop assumption**.
- **Disk failure.** A disk block loses its content as a result of either a head crash or failure during a data transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as tapes, are used to recover from the failure.

Storage Types

- **Volatile storage.** Information stored in volatile storage does not usually survive system crashes. Examples volatile storage: main memory and cache memory. Access to volatile storage is extremely fast.
- **Nonvolatile storage.** Information residing in nonvolatile storage survives system crashes. Example nonvolatile storage is disk and magnetic tapes. Disks are used for online storage, whereas tapes are used for archival storage. Nonvolatile storage is slower than volatile storage. In database systems, disks are used for most nonvolatile storage. Other nonvolatile media are normally used only for backup data.
- **Stable storage.** Information residing in stable storage is never lost.

2. Different Types of Recovery Techniques

2.1. Log-Based Recovery

- The most widely used structure for recording database modifications is the **log**. The log is a sequence of **log records**, recording all the update activities in the database. There are several types of log records.
- We denote various types of log records as:
    1. <Ti start>. Transaction Ti has started (start log record).
    2. <Ti, Xj, V1, V2>. Transaction Ti has performed a write on data item Xj. Xj had value V1 before the write, and will have value V2 after the write (update log record).
    3. <Ti commit>. Transaction Ti has committed (commit log record).
    4. <Ti abort>. Transaction Ti has aborted (abort log record).

2.1.1. Deferred Database Modification

- In this, the modifications done by the transaction are not recorded in the database but **deferred (postponed)** until the transaction commits. However, these modifications are recorded in the log file.

- When a transaction commits, the modifications are applied to the database using the log file. This process is called as re-doing.
- If the system crashes before the transaction completes its execution, or if the transaction aborts, then the information on the log is simply ignored.
- As an example, consider the following; Assume initial values of A=50 and B=100

Transaction(T1)	**Database Before**	**Database After**	**Log**
Start	A=50, B=100	A=50, B=100	<T1, start>
Read(A)	50	50	--
Update A to 500	50	50	<T1,A,50,500>
Read(B)	100	100	---
Update B to 200	100	100	<T1,B,100,200>
Commit		A=500, B=200	<T1, Commit>

Transaction(T1)	**Database Before**	**Database After**	**Log**
Start	A=50, B=100	A=50, B=100	<T1, start>
Read(A)	50	50	--
Update A to 500	50	50	<T1,A,50,500>
Read(B)	100	100	---
Update B to 200	100	100	<T1,B,100,200>
Abort		A=50, B=100	<T1, Abort>

2.1.2.Immediate Database Modification

- In this, the modifications done by the transaction are immediately applied to the database. However, these modifications are first recorded in the log file before applying to the database.
- When a transaction commits, the modifications are made permanent and the log contents are discarded.
- If the system crashes before the transaction completes its execution, or if the transaction aborts, all the modifications done to the database are discarded. This process is called as Un-doing.
- As an example, consider the following; Assume initial values of A=50 and B=100

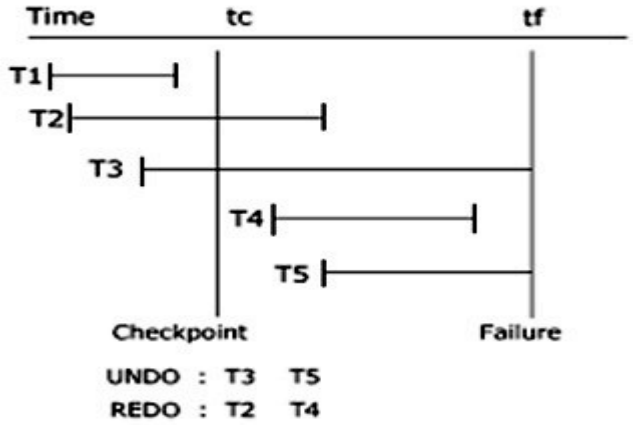
Transaction(T1)	Database Before	Database After	Log
Start	A=50, B=100	A=50, B=100	<T1, start>
Read(A)	50	50	--
Update A to 500	50	500	<T1,A,50,500>
Read(B)	100	100	---
Update B to 200	100	200	<T1,B,100,200>
Commit		A=500, B=200	<T1, Commit>

Transaction(T1)	Database Before	Database After	Log
Start	A=50, B=100	A=50, B=100	<T1, start>
Read(A)	50	50	--
Update A to 500	50	500	<T1,A,50,500>
Read(B)	100	100	---
Update B to 200	100	200	<T1,B,100,200>
Abort		A=50, B=100	<T1, Abort>

- Both *redo* and *undo* operations are idempotent; that is, executing it several times must be equivalent to executing it once.

2.1.3.Check-pointing

- When a system failure occurs, we must use the log to determine those transactions that need to be redone and those that need to be undone. In principle, we need to search the entire log to determine this information.
- There are two major difficulties with this approach:
  1. The search process is time consuming.
  2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.
- To reduce these types of overhead, the system periodically performs **checkpoints.**
- Updating the database at fixed intervals of time is called as check-pointing.
- As an example, consider the following figure.



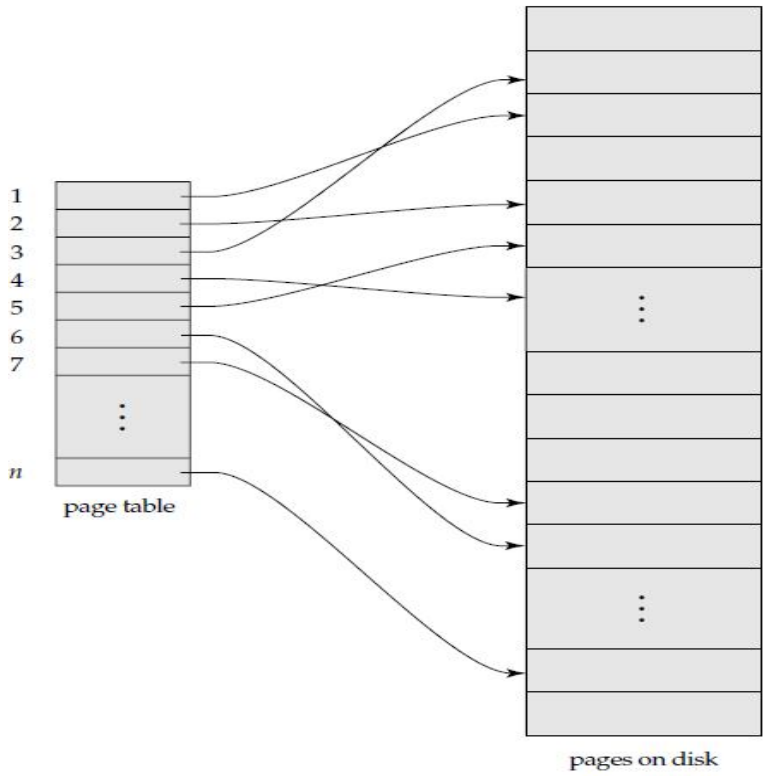
In check pointing, recovery process proceeds as follows.

- As shown in the figure, T1 has committed before check point hence, nothing is done to T1.

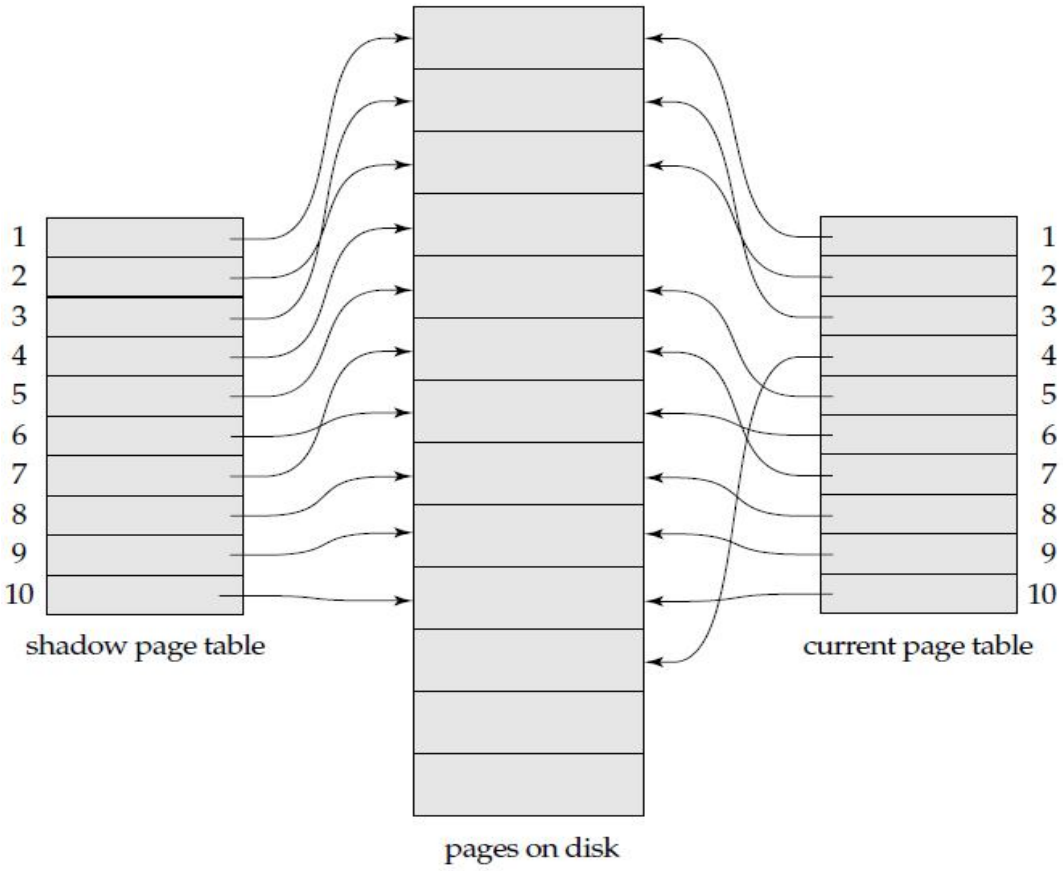
- T2 and T4 have not committed before check point but committed before system failure hence, T2 and T4 are Redone.
- T3 and T5 have not committed before check point as well as before system failure hence, T3 and T5 are Undone.

3. Shadow Paging

- An alternative to log-based crash-recovery techniques is **shadow paging**.
- Shadow paging may require fewer disk accesses than do the log-based methods discussed previously.
- The database is partitioned into some number of fixed-length blocks, which are referred to as **pages** as shown in the figure below.
- The key idea behind the shadow-paging technique is to maintain *two* page tables during the life of a transaction: the **current page table** and the **shadow page table**.



- When the transaction starts, both page tables are identical. The shadow page table is never changed over the duration of the transaction. The current page table may be changed when a transaction performs a write operation.
- All modifications done by the transaction are applied to the current page table only.
- Suppose that the transaction T_j performs a write(X) operation, it modifies the current page table so that the i^{th} entry points to the modified page.



- The above figure shows the shadow and current page tables of some transaction T_j. It also shows that transaction T_j has modified the data item present in 4^{th} page table.

- When a transaction commits before a system crash, the shadow page table is replaced with the current page table, since we may need it for recovery from a crash.
- When a transaction does not commit before a system crash, the current page table is simply ignored and the shadow page table is used as it is.
- The main disadvantage of shadow paging is that it requires the transactions be executed serially.

UNIT VI

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Which of the following is under failure classification? [ ]
   i. Transaction failure
   ii. System crash
   iii. Disk Failure
   iv. Storage failure

A) i and ii B) i , ii and iii

C) i, ii, iii and iv D) ii and iii

2. The most widely used structure for recording database modifications is called _____.

3. Identify crash recovery techniques from the following. [ ]

A) Log-Based B) Check-pointing

C) Shadow- Paging D) All of the above

4. An update log record contains which of the following [ ]

A) Transaction identifier B) Data item identifier

C) Old value and new value D) All of the above

5. What is the effect of the UNDO operation corresponding to a log record where Ti is the transaction, and V1 and V2 are the old and new values respectively of a data location X? [ ]

A) No change to X C) Writes the value V1 to X

B) Writes the value V2 to X D) Sets X to 0

6. Consider the log of transactions below. [ ]

< T0 start >

< T0, S, 100, 120 >

< T0, H, 1, 3 >

Identify the correct actions, which are part of the Undo(T0)

A) H is restored to 3 C) S is set to 120

B) H is restored to 1 D) < T0, S, 100 > log record is written out

7. Identify the options which are true for Immediate-modification scheme.

A) Allows updates of an uncommitted transaction to be made to the buffer, or the disk itself, before the transaction commits. [ ]

B) Update log record must be written after a database item is written.

C) Output of updated blocks to disk storage can take place at any time before or after transaction commit.

D) Performs updates to buffer/disk only at the time of transaction commit

8. Consider the following log sequence: [ ]

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 700, 600>
```

What will be the recovery action by immediate modification recovery?

A) undo T0, redo T1 C) redo T0, redo T1

B) redo T0, undo T1 D) undo T0, undo T1

9. Identify the incorrect statement based on checkpointing. [ ]

A) All updates are stopped while doing check pointing.

B) Continue scanning backwards till a record < Ti start > is found for every transaction Ti in L.

C) Scan backwards from end of log to find the most recent < checkpoint L > record.

D) Scan from starting of log to find the most recent < checkpoint L > record.

10. What is the effect of the UNDO operation corresponding to a log record < Ti , Y, S, K > where Ti is the transaction, and S and K are the old and new values respectively of a data location Y? [ ]

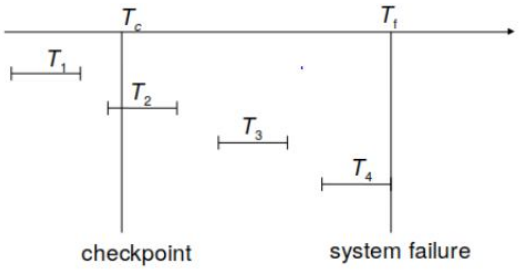
A) No change to Y C) Writes the value S to Y

B) Writes the value K to Y D) Sets Y to 0.

SECTION-B

Descriptive Questions

1. Explain the concept of failure classification.
2. Distinguish between different Storage Mechanisms
3. Explain different types of Recovery Techniques.
4. Distinguish between immediate and deferred database modification (update).
5. Illustrate check pointing with an example.
6. Summarize the importance of shadow paging.
7. Consider the following state of transactions:



and the statements below:

1. T1 can be ignored.
2. T2 and T3 redone
3. T4 undone
4. T4 redone

Mark the correct group of statements from the options below.

A) 1), 2), 4)

B) 1), 2), 3)

C) only 1) and 2) but not 3)

D) only 2) and 3) but not 1)

8. Consider the log of transactions given below and answer the Q. No-6 and Q. No-7:

< T2 start >

< T2, H, 18, 20 >

< T3 start >

< checkpoint {T2, T3} >

< T3 commit >

< T4 start >

< T4, G, 6, 7 >

< T2, Y, 12 >

< T2 abort >

Suppose there is a crash after the record < T2 abort >.

Identify the correct statement(s) from the Redo phase

A) The undo list initially contains T2, T3

B) The undo list initially contains T2, T3, T4

C) T3 is removed from undo list after some steps

D) T2 is removed from undo list after some steps.

9. Identify the incorrect statement(s) based on the Undo phase

A) The undo list at the start of the undo phase contains T2, T4

B) < T4, G, 6 > log record is written out

C) The undo list at the start of the undo phase contains T2

D) < T4, abort > log record is written out