1. (a) Define: DBMS. Discuss the main characteristics of the database approach and how it differs from traditional file system. (10)

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

▪ Self-Describing Nature of a Database System

- the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- The information stored in the catalog is called meta-data, and it describes the structure of the primary database

▪ Insulation between Programs and Data, and Data Abstraction

- Program data independence: The structure of data files is stored in the DBMS catalog separately from the access programs.
- Program-operation independence: User application programs can operate on the data by invoking these operations
- Data abstraction: The characteristic that allows program-data independence and program operation independence
- Data model: is a type of data abstraction that is used to provide this conceptual representation

▪ Support of Multiple Views of the Data

- A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored

▪ Sharing of Data and Multiuser Transaction Processing

- DBMS allows multiple users to access the database at the same time
- The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct

1. (b) Discuss the main activities of actors on the scene and workers behind the scene. (10)

(i) Actors on the Scene:

- the people whose jobs involve the day-to-day use of a large database

i. Database Administrators: responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed

ii. Database Designers: responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.

- develop views of the database that meet the data and processing requirements of these groups.

iii. End Users: the people whose jobs require access to the database for querying, updating, and generating reports

- several categories of end users:
  - Casual end users: occasionally access the database, but they may need different information each time (middle or higher-level managers)
  - Naive or parametric end users: constantly querying and updating the database (bank tellers, reservation agents)
  - Sophisticated end users: include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
  - Standalone users: maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

iv. System Analysts and Application Programmers

- System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.

Workers behind the scene:

- they include the following categories:

i. DBMS system designers and implementers: design and implement the DBMS modules and interfaces as a software package

ii. Tool developers: the software packages that facilitate database modeling and design, database system design, and improved performance.

iii. Operators and maintenance personnel: responsible for the actual running and maintenance of the hardware and software environment for the database system

2. (a) With a neat diagram, explain the three-schema architecture of DBMS. **(6)**

- Three important characteristics of the database approach are:

(1) use of a catalog to store the database description (schema) so as to make it self-describing,

(2) insulation of programs and data (program-data and program-operation independence), and

(3) support of multiple user views.

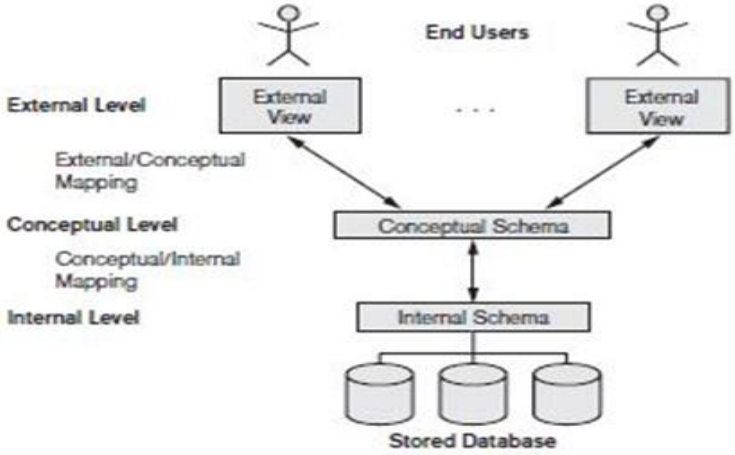
- three-schema architecture: specify an architecture for database systems that was proposed to help achieve and visualize these characteristics

- The goal of the three-schema architecture is to separate the user applications from the physical database.
- In this architecture, schemas can be defined at the following three levels:

i. internal level: has internal schema - physical storage structure

ii. conceptual level: has a conceptual schema - structure of the whole database

iii. external or view level: includes a number of external schemas or user views.



- it is a convenient tool with which the user can visualize the schema levels in a database system
- The processes of transforming requests and results between levels are called mappings

2. (b) Discuss database language and interfaces. (8)

- The DBMS must provide appropriate languages and interfaces for each category of users
- DBMS Languages

i. Data Definition Language (DDL): used by the DBA and by database designers to define conceptual and internal schemas

ii. Storage Definition Language (SDL): used to specify the internal schema

iii. View Definition Language (VDL): specify user views and their mappings to the conceptual schema

- The DBMS provides a set of operations or a language called the Data Manipulation Language (DML) to manipulate the database
- There are two main types of DMLs:
- A high level non-procedural language to specify complex database operations (Example: SQL)

(Eg. SQL)

- follows row-at-a-time processing

ii. A low-level or procedural DML must be embedded in a general-purpose programming language. (Eg. PL/SQL)

- DBMS Interfaces: User-friendly interfaces provided by a DBMS may include the following:
- *Menu-Based Interfaces for Web Clients or Browsing (Pull down menus)*
  - Menus do away with the need to memorize the specific commands and syntax of a query language

- *Forms-Based Interfaces:* displays a form to each user, can fill out all of the form entries to insert new data, or they can fill out only certain entries
- *Graphical User Interfaces:* displays a schema to the user in diagrammatic form
- utilize both menus and forms
- *Natural Language Interfaces:* accept requests written in English or some other language and attempt to understand them
- *Speech Input and Output:* Limited use of speech as an input query and speech as an answer to a question or result of a request (Eg. Telephone directory information)
- The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries.

2. (c) What is entity type, entity set? Explain the difference between entity, entity type and entity set. (6)

- The basic object that the ER model represents is an entity, which is a thing in the real world with an independent existence.
- An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.
- The collection of all entities of a particular entity type in the database at any point in time is called an entity set; the entity set is usually referred to using the same name as the entity type.

3. (a) Discuss the characteristics of relations. (10)

- ***Ordering of Tuples in a Relation***: A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order.
- ***Ordering of Values within a Tuple and an Alternative Definition of a Relation***: an n-tuple is an ordered list of n values, so the ordering of values in a tuple - and hence of attributes in a relation schema - is important. However, at a more abstract level, the order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.
- ***Values and NULLs in the Tuples***: Each value in a tuple is an atomic value; that is, it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed.
- ***Interpretation (Meaning) of a Relation***: The relation schema can be interpreted as a declaration or a type of assertion.

3. (b) Discuss the entity integrity and referential integrity in detail. (10)

- The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples.

- The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
- To define referential integrity more formally, first we define the concept of a foreign key.
- The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R1 and R2. A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R_2; the attributes FK are said to reference or refer to the relation R_2.
2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL. In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 references or refers to the tuple t_2.

- R_1 is called the referencing relation and R_2 is the referenced relation.
- If these two conditions hold, a referential integrity constraint from R_1 to R_2 is said to hold

4. (a) Explain the following: (12)
i) Select ii) Project iii) Rename iv) Cartesian Product v) Division operation

(i) SELECT (σ)

- used to choose a subset of the tuples from a relation that satisfies a selection condition
- the SELECT operation is denoted by: σ <selection condition> (R)

o Eg.: σ salary>30000 (EMPLOYEE)

- The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

<attribute name> <comparison op> <constant value> or
<attribute name> <comparison op> <attribute name>
(Eg.) $\sigma_{(Dno=4\ AND\ Salary>25000)\ OR\ (Dno=5\ AND\ Salary>30000)}$(EMPLOYEE)

(ii) PROJECT (π)

- selects certain columns from the table and discards the other columns
- the result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations:
  - one has the needed columns (attributes) and contains the result of the operation, and
  - the other contains the discarded columns
- The general form of the PROJECT operation is: $\pi_{<attribute\ list>}(R)$

o Eg.: $\pi_{Lname, Fname, Salary}$ (EMPLOYEE)

- The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of

distinct tuples (duplicate elimination)

- $\pi_{Sex, Salary}$(EMPLOYEE) is equivalent to select DISTINCT sex, salary from EMPLOYEE;

iii. Rename:

- It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.
- We can also use this technique to rename the attributes in the intermediate and result relations.
- To rename the attributes in a relation, we simply list the new attribute names in parentheses. (Example: TEMP ← $\sigma_{Dno=5}$(EMPLOYEE)
- We can also define a formal RENAME operation - which can rename either the relation name or the attribute names, or both - as a unary operator.
- The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

  ρS(B1, B2, ..., Bn)(R) or ρS(R) or ρ(B1, B2, ..., Bn)(R)

iv. Cartesian Product (also known as CROSS PRODUCT or CROSS JOIN)

- a binary set operation
- denoted by X
- the relations on which it is applied do not have to be union compatible
- this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set)
- the result of $R(A_1, A_2, ..., A_n) \times S(B_1, B_2, ..., B_m)$ is a relation Q with degree n+m attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$, in that order
- The resulting relation Q has one tuple for each combination of tuples
- if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R × S will have $n_R * n_S$ tuples

v. Division Operator: (denoted by ÷, applied to two relations)

- R(Z) ÷ S(X) where X subset Z. Let Y = Z – X, let Y be the set of attributes of R that are not attributes of S
- The result of division is a relation T(Y) that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with $t_R[X] = t_S$ for every tuple t_S in S
- useful for a special kind of query that sometimes occurs in database applications
- For a tuple t to appear in the result T of the DIVISION operation, the values in t must appear in R in combination with every tuple in S
- Example: identify all clients who have viewed all properties with three rooms

4. (b) Consider the following relational schema and answer the following queries using relational algebra: (8)

DEPARTMENT (Dnumber, Dname, MgrSSN, MgrStartDate)
PROJECT (Pnumber, Pname, Plocation, Dnumber)
EMPLOYEE (SSN, Name, Bdate, Addr, Sex, Salary, SuperSSN, DNumber)
DEPENDENT (ESSN, Dependent_name, Sex, Bdate, Relationship)

i) Retrieve the name and address of all employees who work for account department.
ii) Retrieve the department number, number of employees and their average salary.
iii) Retrieve the name and salary of the manager of each department.
iv) List the names of mangers who have atleast one dependent.

5. (a) Define SQL data definition and date types in brief and explain with example, Aggregate functions. (10)

- The set of relations in a database must be specified to the system by means of a data definition language (DDL).
- It allows specification of not only a set of relations, but also information about each relation, including :
- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk
- Basic Types: The SQL standard supports a variety of built-in types, including:
  - Char(n): A fixed-length character string with user-specified length n
  - varchar(n): A variable-length character string with user-specified maximum length n
  - int: An integer (a finite subset of the integers that is machine dependent)
  - smallint: A small integer (a machine-dependent subset of the integer type)
  - numeric(p, d): A fixed-point number with user-specified precision.
  - real, double precision: Floating-point and double-precision floating-point numbers with machine-dependent precision
  - float(n): A floating-point number, with precision of at least n digits
- The general form of the create table command is:
- create table r (A1 D1, A2 D2, . . . , An Dn, <integrity-constraint1>, . . . , <integrity-constraintk);
- **Data Types:**
- the SQL standard supports several data types relating to dates and times:
- date: A calendar date containing a (four-digit) year, month, and day of the month
- time: The time of day, in hours, minutes, and seconds
- timestamp: A combination of date and time
- Default Values
- SQL allows a default value to be specified for an attribute as illustrated by the following create table statement:
- Large-Object Types
- SQL provides large-object (LOB) data types for character data (clob) and binary data (blob).
  - book_review clob(10KB)

- image blob(10MB)
- movie blob(2GB)

Aggregate Functions:

- These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include SUM,
- AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.
- We can define an AGGREGATE FUNCTION operation, using the symbol ℑ (pronounced script F), to specify these types of requests as follows:
  <grouping attributes> ℑ <function list> (R)
  - where <grouping attributes> is a list of attributes of the relation specified in R, and
  - <function list> is a list of (<function> <attribute>) pairs

5. (b) Explain the different clauses of SELECT – FROM – WHERE – GROUP – HAVING with an example for each. **(10)**

i. The select command is used to retrieve a specified columns or all the columns from the table name or from the system generated object.

Eg. Select sysdate from dual;

ii. i. The select command is used to retrieve a specified columns or all the columns from the table name

Eg. Select * from student;

Select USN, studname from student;

iii. The select command with where clause is used to retrieve the columns from the table with the condition specified in the where clause.

Eg. Select * from instructor

Where salary > 50000;

iv. The select command offers to summarise the data based on some specific attribute. It is noted that the group by clause should have any one of the aggregate functions supported by the RDBMS.

Eg. Select deptid, sum(dalary) from instructor

Group by deptname;

v. The select command with having clause can be used to group the data for a specified condition.

Eg. Select deptid, avg(salary) from instructor

Group by deptname

Having count(deptid) > 2;

6. (a) Consider the following tables and write the SQL for the following: **(10)**

PROJECT (Pnumber, Pname, Plocation, Dnumber)
EMPLOYEE (SSN, Name, Bdate, Addr, Sex, Salary, SuperSSN, Dno)
WORKS_ON(Essn, Pno, Hours)
DEPENDENT (Essn, Dependent_name, Sex, Bdata, Relationship)

i) Retrieve the name and address of all employees who work for the 'Research Department'.

ii) For each employee, retrieve the employee's first and last name and first and last name of his or her immediate supervisor.
iii) Retrieve list of employees and the projects they are working on, ordered by department and within each department, ordered alphabetically by last name, first name.
iv) Retrieve all employees whose address is in Houstan, Texas.
v) Find the sum of the salaries of all employees, the maximum salary, the minimum salary and the average salary.

Solution:

i. select fname, lname, address
from employee
where dno in
(select dnumber from department where dname='research');

ii. select e.fname, e.lname, s.fname, s.lname from employee e s
where e.superssn=s.ssn;

iii. select dname, lname, fname, pname from department, employee, works_on, project
where dnumber=dno and ssn=essn and pno=pnumber
order bydname, lname;

iv. select fname, lname from employee
where address like '%houston,tx%;

v. select sum(salary), max(salary), min(salary), avg(salary) from employee;

6. (b) Discuss database programming issues and techniques. (10)

- Approaches to Database Programming
- Several techniques exist for including database interactions in application programs.
- The main approaches for database programming are the following:

i. Embedding database commands in a general-purpose programming language: database statements are embedded into the host programming language, but they are identified by a special prefix (Example: EXEC SQL)

ii. Using a library of database functions: A library of functions is made available to the host programming language for database calls, such as connect to a database, execute a query

iii. Designing a brand-new language: A database programming language is designed from scratch to be compatible with the database model and query language, including looping and conditional structures. (Example: PL/SQL)

- Impedance Mismatch
- refer to the problems that occur because of differences between the database model and the programming language model
- Typical Sequence of Interaction in Database Programming
- When writing such a program, a common sequence of interaction is the following:
- the program must first establish or open a connection to the database server (connection establishment)

- the program can interact with the database by submitting queries, updates, and other database commands (Query execution)
- When the program no longer needs access to a particular database, it should terminate or close the connection to the database (connection termination)

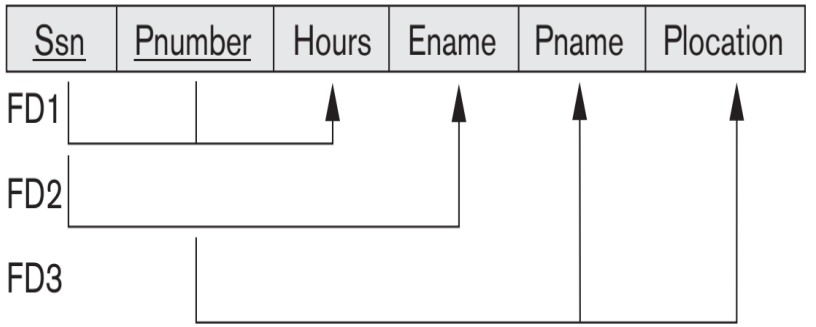
7. (a) Define functional dependency and explain informal design guidelines for relation schemas. (10)

Functional Dependency:

- a formal tool for analysis of relational schemas that enables to detect and describe some of the problems in precise terms
- The single most important concept in relational schema design theory is that of a functional dependency

A functional dependency (denoted by X → Y) between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$

- the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component.
- there is a functional dependency from X to Y, or that Y is functionally dependent on X.



- {Ssn, Pnumber} → Hours
- Ssn → Ename
- Pnumber → Pname, Plocation
- A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R.

- an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R
- Given a populated relation, one cannot determine which FDs hold and which do not unless the meaning of and the relationships among the attributes are known
- One cannot guarantee its existence until the meaning of the corresponding attributes is clearly understood
- numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F

- Those other dependencies can be inferred or deduced from the FDs in F

Informal Design Guidelines:

- Four informal guidelines that may be used as measures to determine the quality of relation schema design:

1. Making sure that the semantics of the attributes is clear in the schema
2. Reducing the redundant information in tuples
3. Reducing the NULL values in tuples
4. Disallowing the possibility of generating spurious tuples

- These measures are not always independent of one another

(i) Imparting Clear Semantics to Attributes in Relations

- Attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them
- The ease with which the meaning of a relation's attributes can be explained is an informal measure of how well the relation is designed
- Guideline-1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. If a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

(ii) Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations
- Grouping attributes into relation schemas has a significant effect on storage space
- Storing natural joins of base relations leads to an additional problem referred to as update anomalies.
- These can be classified into:
  i. insertion anomalies
  ii. deletion anomalies and
  iii. modification anomalies
- Guideline-2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

(iii) NULL Values in Tuples

- In some schema designs, it may be required to group many attributes together into a "fat" relation.
- If many of the attributes do not apply to all tuples in the relation, it will end up with many NULLs in those tuples
- this leads to waste the space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level
- NULL values have greater impact with aggregate functions

- if NULL values are present, the results may become unpredictable during the comparison operations such as select and join
- NULLs can have multiple interpretations, such as the following:
  - The attribute does not apply to this tuple (Not applicable)
  - The attribute value for this tuple is unknown
  - The value is known but absent
- Having the same representation for all NULLs compromises the different meanings they may have
- Guideline-3: As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation

(iv) Generation of Spurious Tuples

- The natural join on two relations produce a new relation, which contains the tuples based on the specified condition
- If an attempt is being made using a natural join operation on two relations, the results produces many more tuples than the original set of tuples in the relation
- The additional tuples generated by such join operation is called spurious tuples
- Guideline 4: Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

7. (b) What is normalization? Explain 1NF, 2NF and 3NF with example for each. (10)

- Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of:
  - minimizing redundancy and
  - minimizing the insertion, deletion, and update anomalies
- Definition: The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized

(i) First Normal Form (1NF)

Definition: The relation is said to be in 1NF if all the attributes in a relation must have atomic domains

(ii) Second Normal Form (2NF)

Definition: A relation schema R is in 2NF if every non-prime attribute A in R is fully functionally dependent on the primary key of R

(iii) Third Normal Form (3NF)

A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X → A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R

8. (a) What is stored procedure? Write the syntax for creating stored procedure. (10)

- Procedures and functions allow "business logic" to be stored in the database, and executed from SQL statements
- SQL allows the definition of functions, procedures, and methods. These can be defined either by the procedural component of SQL, or by an external programming language such as Java, C, or C++

Procedure Declaration:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
   Execution section;
[EXCEPTION
  Exception section ]
END;
```

- SQL permits more than one procedure of the same name, so long as the number of arguments of the procedures with the same name is different. The name, along with the number of arguments, is used to identify the procedure
- It allows the language constructs like for…loop, while…loop, loop… exit when, if..then, if…then…else, if…then…elsif…else and case…when.

8. (b) What is a trigger? Explain DML trigger, with an example. (10)

- A trigger is a statement that the system executes automatically as a side effect of a modification to the database.
- Triggers are stored programs, which are automatically executed or fired when some events occur
- Two requirements to design a trigger mechanism:

(i) Specify when a trigger is to be executed.

This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.

(ii) Specify the actions to be taken when the trigger executes

- Need for Triggers
- used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL
- useful mechanisms for alerting humans for starting certain tasks automatically when certain conditions are met
- Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_name
[FOR EACH ROW]
[WHEN condition]
DECLARE
        declaration statements
BEGIN
        executable statements
EXCEPTION
        exception_handling statements
```

END;

- When Not to Use Triggers
- it can be used as an alternate to "on delete cascade"
  - this would require more work to implement, it would be much harder for a database user to understand the set of constraints implemented in the database
- triggers can be used to maintain materialized views
  - It can be replaced with triggers by using insert, delete or update of column(s)
- Triggers have been used for maintaining copies, or replicas, of databases
- unintended execution of the triggered action when data are loaded from a backup copy, or when database updates at a site are replicated on a backup site
- Triggers should be written with great care, since a trigger error detected at runtime causes the failure of the action statement that set off the trigger

9. (a) What is transaction? In what ways it is different from an ordinary program. (10)

- Collections of operations that form a single logical unit of work
- a unit of program execution that accesses and possibly updates various data items
- The actions that can be executed by a transaction include reads and writes of database objects, whereas actions in an ordinary program could involve user input, access to network devices, user interface drawing, etc
- A transaction needs to be either committed or rolled back depends on the state of the transaction, whereas in an ordinary program it need not be done. The file processing and data access can be handled easily inside the program itself.
- A transaction needs to satisfy the ACID properties, whereas it is not a requirement for an ordinary program.
- A transaction can have states whereas for an ordinary program there may be states like design, compilation and running states.

9. (b) Explain the database recovery technique based on different update. (10)

- when a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.
- it is not a requirement that every update be applied immediately to disk; it is just possible that some updates are applied to disk before the transaction commits.

(i) No-UNDO / REDO Recovery based on deferred update:

- only REDO-type log entries are needed in the log, which include the new value (AFIM) of the item written by a write operation. The UNDO-

type log entries are not needed since no undoing of operations will be required during recovery.

- Procedure RDU_M (NO-UNDO/REDO with checkpoints). Use two lists of transactions maintained by the system: the committed transactions T since the last checkpoint (commit list), and the active transactions T (active list).
- REDO all the WRITE operations of the committed transactions from the log, in the order in which they were written into the log. The transactions that are active and did not commit are effectively canceled and must be resubmitted.
- The REDO procedure is defined as follows:
- Procedure REDO (WRITE_OP). Redoing a write_item operation WRITE_OP consists of examining its log entry [write_item, T, X, new_value] and setting the value of item X in the database to new_value, which is the after image (AFIM).

(ii) Recovery technique based on immediate update:

- In these techniques, when a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.
- the UNDO-type log entries, which include the old value (BFIM) of the
- item, must be stored in the log. Because UNDO can be needed during recovery, these methods follow a steal strategy for deciding when updated main memory buffers can be written back to disk.
- Procedure RIU_M (UNDO/REDO with checkpoints).

1. Use two lists of transactions maintained by the system: the committed transactions since the last checkpoint and the active transactions.
2. Undo all the write_item operations of the active (uncommitted) transactions, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.
3. Redo all the write_item operations of the committed transactions from the log, in the order in which they were written into the log, using the REDO procedure defined earlier.

- The UNDO procedure is defined as follows:

Procedure UNDO (WRITE_OP). Undoing a write_item operation write_op consists of examining its log entry [write_item, T, X, old_value, new_value] and setting the value of item X in the database to old_value, which is the before image (BFIM). Undoing a number of write_item operations from one or more transactions from the log must proceed in the reverse order from the order in which the operations were written in the log.

10. (a) Explain i) ACID properties ii) Strict two phase locking. (10)

(i) ACID properties of the transactions:

i. Atomicity: Either all operations of the transaction are reflected properly in the database, or none are. (“All-or-Nothing property”)
ii. Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

iii. Isolation: Even though multiple transactions may execute concurrently, the system guarantees that, 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.

iv. Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

(ii) Strict two-phase locking protocol: a modification of two phase locking to avoid cascading rollbacks

- requires that:
  - locking be two phase
  - all exclusive-mode locks taken by a transaction be held until that transaction commits
- rigorous two-phase locking protocol: Another variant of two-phase locking which requires that all locks be held until the transaction commits
- refinement of the basic two-phase locking protocol, in which lock conversions provide a mechanism for
  - upgrading a shared lock to an exclusive lock; upgrading can take place in only the growing phase and
  - downgrading an exclusive lock to a shared lock; downgrading can take place in only the shrinking phase

First Phase	Second Phase
○ can acquire a lock-S on item ○ can acquire a lock-X on item ○ can convert a lock-S to a lock-X (upgrade)	○ can release a lock-S ○ can release a lock-X ○ can convert a lock-X to a lock-S (downgrade)

10. (b) Discuss deadlock prevention protocols. (10)

- Deadlock Prevention:
- There are two approaches to deadlock prevention
  - ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together (called as no circular wait)
- performs transaction rollback instead of waiting for a lock
  - Another approach: impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering
- A variation of this approach is to use a total order of data items
- use pre-emption and transaction rollbacks
- In pre-emption, when a transaction T_j requests a lock that transaction T_i holds, the lock granted to T_i may be pre-empted by rolling back of T_i, and granting of the lock to T_j
- To control the pre-emption, we assign a unique timestamp
- The system uses these timestamps only to decide whether a transaction should wait or roll back

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

(i) The wait–die scheme: is a non-preemptive technique

- When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (i.e T_i is older than T_j). Otherwise, T_i is rolled back (dies).

(ii) The wound–wait scheme is a pre-emptive technique

- 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 T_j (that is, Ti is younger than Tj). Otherwise, Tj is rolled back (Tj is wounded by Ti).

- Another simple approach to deadlock prevention is based on lock timeouts
  - a transaction that has requested a lock waits for at most a specified amount of time
  - If the lock has not been granted within that time, the transaction is said to time out, and it rolls itself back and restarts
  - If there was in fact a deadlock, one or more transactions involved in the deadlock will time out and roll back, allowing the others to proceed
  - The timeout scheme is particularly easy to implement, and works well if transactions are short and if long waits are likely to be due to deadlocks
  - it is hard to decide how long a transaction must wait before timing out
    - Too long a wait results in unnecessary delays once a deadlock has occurred.
    - Too short a wait results in transaction rollback even when there is no deadlock, leading to wasted resources.