Model Question Paper- II with effect from 2022

CBCS SCHEME

Fourth Semester B.E Degree Examination 2024-25

Database Management System (BCS403)

TIME: 03 Hours *Max.Marks:100*

1. *Note: Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE***
2. *M: Marks, L: Bloom's level, C: Course outcomes.*

		Module - 1	M	L	C
Q.1	**a**	What is a Database? Explain the three schema architecture with neat diagram.	**8**	**L2**	**CO1**
	b	What are the advantages of using DBMS approach? Explain	**8**	**L2**	**CO1**
	c	Explain the following terms. 1. Data Dictionary 2. Weak Entity	**4**	**L2**	**CO1**
		OR			
Q.2	**a**	Explain the categories of Data Models.	**8**	**L2**	**CO1**
	b	Explain the component modules of DBMS & their interactions with diagram.	**8**	**L2**	**CO1**
	c	What are the responsibilities of DBA & database designers?	**4**	**L2**	**CO1**
		Module - 2			
Q.3	**a**	Explain the different types of update operations on relational database. How basic operation deals with constraint violation.	**6**	**L2**	**CO2**
	b	Explain Unary relational operations with examples.	**6**	**L2**	**CO2**
	c	What is an Integrity Constraint? Explain the importance of Referential Integrity Constraint.	**8**	**L2**	**CO2**
		OR			
Q.4	**a**	Explain the following relational algebra operation. JOIN, DIFFERENCE, SELECT, UNION	**10**	**L3**	**CO2**
	b	Discuss the E.R to Relational mapping algorithm with example for each step.	**6**	**L3**	**CO2**
	c	Explain the relational algebra operation for set theory with examples.	**4**	**L2**	**CO2**
		Module - 3			
Q.5	**a**	Illustrate insert, delete, update, alter & drop commands in SQL.	**6**	**L4**	**CO3**

Model Question Paper- II with effect from 2022

	b	Explain informal design guidelines for relational schema design.	**4**	**L2**	**CO3**
	c	What is Functional dependency? Explain the inference rules for functional dependency with proof.	**10**	**L3**	**CO4**
		OR			
Q.6	**a**	Consider two sets of functional dependency. F={A→C, AC→D, E→AD, E→H} E= {A→CD, E→AH}. Are they Equivalent?	**10**	**L3**	**CO4**
	b	Explain the types of update anomalies in SQL with an example.	**10**	**L2**	**CO3**
		Module - 4			
Q.7	**a**	Demonstrate transaction states & additional operations.	**10**	**L3**	**CO4**
	b	Demonstrate working of Assertion & Triggers in database? Explain with an example.	**10**	**L2**	**CO3**
		OR			
Q.8	**a**	Demonstrate the System Log in database transaction.	**6**	**L2**	**CO4**
	b	Discuss the ACID properties of database transaction.	**4**	**L2**	**CO4**
	c	Explain stored procedure language in SQL with an example.	**10**	**L2**	**CO3**
		Module - 5			
Q.9	**a**	Explain the Two phase locking protocol used for concurrency control.	**8**	**L3**	**CO5**
	b	Define Schedule? Illustrate with an example.	**4**	**L2**	**CO5**
	c.	Why Concurrency control is needed? Demonstrate with an example.	**8**	**L3**	**CO5**
		OR			
Q.10	**a**	What is NOSQL? Explain the CAP theorem.	**6**	**L2**	**CO5**
	b	What are document based NOSQL systems? basic operations CRUD in MongoDB.	**8**	**L2**	**CO5**
	c	What is NOSQL Graph database? Explain Neo4j.	**6**	**L2**	**CO5**

Model Question Paper- I with effect from 2022

CBCS SCHEME

Fourth Semester B.E Degree Examination 2024-25

Database Management Systems (BCS403)

TIME: 03 Hours *Max.Marks:100*

1. *Note: Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE***
2. *M: Marks, L: Bloom's level, C: Course outcomes.*

		Module - 1	M	L	C
Q.1	**a**	Explain the types of end users with examples.	**8**	**L2**	**CO1**
	b	What are the advantages of using DBMS? Explain.	**8**	**L2**	**CO1**
	c	Describe the characteristics of database.	**4**	**L2**	**CO1**
		OR			
Q.2	**a**	Explain three schema architecture. Why mappings b/w schema levels are required?	**8**	**L2**	**CO1**
	b	Explain the different types of attributes in ER model.	**8**	**L2**	**CO1**
	c	Explain the following. 1. Cardinality Ratio 2. Weal Entity	**4**	**L2**	**CO1**
		Module - 2			
Q.3	**a**	Explain the different Relational Model constraints.	**6**	**L2**	**CO2**
	b	Demonstrate the concepts of Generalization & Specialization with examples.	**6**	**L2**	**CO2**
	c	Explain Entity Integrity Constraint & Referential Integrity Constraints? Why each of these is important in a database.	**8**	**L2**	**CO2**
		OR			

Model Question Paper- I with effect from 2022

Q.4	**a**	Consider the Sailors-Boats-Reserves DB described **s (sid, sname, rating, age)** **b (bid, bname, color)** **r (sid, bid, date)** Write each of the following queries in SQL. 1. Find the colors of boats reserved by Alber. 2. Find all sailor ids of sailors who have a rating of at least 8 or reserved boat 103. 3. Find the names of sailors who have not reserved a boat whose name contains the string "storm". Order the names in ascending order. 4. Find the sailor ids of sailors with age over 20 who have not reserved a boat whose name includes the string "thunder".	**10**	**L3**	**CO2**
	b	Discuss the Equijoin & Natural Join with suitable example.	**6**	**L3**	**CO2**
	c	Explain the relational algebra operation for set theory with examples.	**4**	**L2**	**CO2**
		Module - 3			
Q.5	**a**	Explain the Cursor & its properties in embedded SQL with an example.	**6**	**L2**	**CO3**
	b	What is a Normalization? Explain the 1NF, 2NF & 3NF with examples.	**10**	**L2**	**CO4**
	c	Explain informal design guidelines for relational schema design.	**4**	**L2**	**CO3**
		OR			
Q.6	**a**	What is Functional Dependency? Write algorithm to find minimal cover for set of Functional Dependency. Construct the minimal cover m for set of functional dependency. E={ B→A, D→A, AB→D}	**10**	**L2**	**CO4**
	b	Explain the types of update anomalies in SQL with an example.	**10**	**L4**	**CO3**
		Module - 4			
Q.7	**a**	Demonstrate the Database Transaction with transaction diagram.	**10**	**L2**	**CO4**
	b	Demonstrate working of Assertion & Triggers in SQL? Explain with an example.	**10**	**L3**	**CO3**
		OR			
Q.8	**a**	Demonstrate the System Log in database transaction.	**6**	**L2**	**CO4**
	b	Demonstrate the ACID properties of database transaction.	**4**	**L2**	**CO4**
	c	Explain stored procedure language in SQL with an example.	**10**	**L2**	**CO3**

Model Question Paper- I with effect from 2022

		Module - 5			
Q.9	**a**	Demonstrate the Two phase locking protocol used for concurrency control.	**8**	**L3**	**CO5**
	b	Demonstrate the Concurrency control based on Timestamp ordering.	**4**	**L2**	**CO5**
	c.	Why Concurrency control is needed? Demonstrate with an example.	**8**	**L3**	**CO5**
		OR			
Q.10	**a**	What is NOSQL? Explain the CAP theorem.	**6**	**L2**	**CO5**
	b	What are document based NOSQL systems? Explain basic operations CRUD in MongoDB.	**8**	**L2**	**CO5**
	c	What is NOSQL Graph database? Explain Neo4j.	**6**	**L2**	**CO5**

M T W T F S S

Date :

DATABASE MANAGEMENT SYSTEM (BCS403)

MODEL QESTION PAPER-2024

1. What is a Database? Explain the three schema architecture with neat diagram.

Defining a database involves specifying the data types, structure, and constraints for the data to be stored in the database.

Three-Schema Architecture

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).

3) Support of multiple user views.

* Three-schema architecture, that was proposed to help achieve and visualize these characteristics.

* It Defines DBMS Schems at three levels:

M T W T F S S

Date :

To Complete more Syllabus in short time, create a study plan, focus on key concepts, eliminate distractions, use active learning techniques, and stay organized, while seeking help when needed and getting enough sleep.

STUDY TIP

Internal Schema:

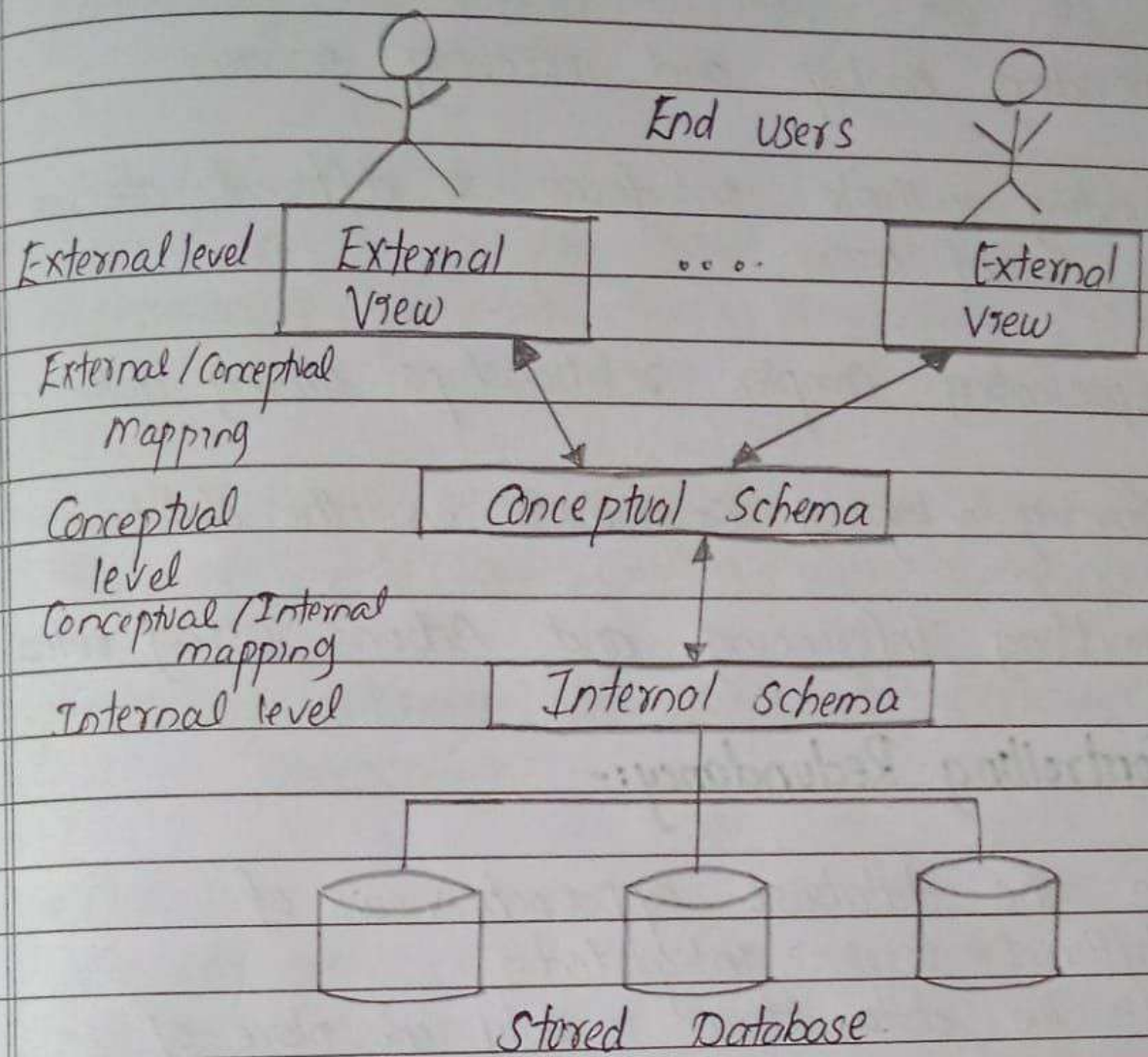
The internal level has an internal schema, which describes the physical storage structure of the database.
The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

Conceptual Schema.

which describes the structure of the whole database for a community of users. The Conceptual schema hides the details of physical storage structure and concentrates on describing entities, data types, relationships, user operations, and Constraints.

External Schema

The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from the user group.



b) What are the advantages of using DBMS approach? Explain.

1. Controlling redundancy in data storage and in development and maintenance efforts.
2. Restricting unauthorized access to data.
3. providing persistent storage for program objects
4. providing storage structure for efficient Query processing.

5. providing backup and recovery services.
6. providing multiple interfaces to different classes of users.
7. Representing complex relationships among data.
8. Enforcing integrity constraints on the database
9. permitting Inferencing and Actions using rules.

Controlling Redundancy:-

- In the database approach, views of different users -> integrated.
- All the data stored in only one place of the database
- This ensures consistency & saves storage space.

Restricting Unauthorized Access:-

- when multiple users share a large database, the type of access operation must be controlled.
- DBMS must provides security and authorization subsystem.
- DBA -> creates accounts and specifies account restrictions.
- Parametric users -> allowed to access database only through canned transactions.

M T W T F S S

COMPASS

Date :

providing persistent storage for Program object:

Traditional file system:

- must be explicitly stored in seperate permanent files -> involves conversion.

DBMS

- But DBMS recognizes data structures of program language like Java, C++ -> automatically converts.

providing storage structures for Effcient Query processing

- Database system must provide capabilities for efficiently executing queries and updates.
- Since database is stored on disk, DBMS must provide specialized data structures (indexes) to speed up disk search.
- Query processing and optimization module -> responsible for efficient query execution.

providing Backup and Recovery

- The backup and recovery subsystem of DBMS -> responsible for recovery -> in case of hardware or software failures.

Eg:- If the computer crashes during a complex transaction, the recovery subsystem -> responsible for ensuring that transaction resumes from where it is interrupted or atleast restore

to the state it was before transaction started executing.

Providing Multiple User Interfaces

- Multiple users → different levels of technical knowledge → so DBMS should provide a variety of user interfaces
- form-style interfaces and menu-driven interfaces → Graphical User Interfaces (GUIs).

Representing Complex Relationship among Data

- A database may have variety of data → interrelated in many ways.
- DBMS must be capable of
  - → Representing complex relationship among data.
  - → Retrieve and update related data easily and efficiently.

Enforcing Integrity Constraints

- Simplest type of integrity contraint → Specifying data types for each data items.
- Another type of constraint → uniqueness of data item values.
- Responsibility of Database designer → identifying integrity constraints during database design.

M T W T F S S

COMPASS

Date :

c) Explain the following terms.

1. Data Dictionary

In a database management system (DBMS), a data dictionary can be defined as a component that stores a collection of names, definitions, and attributes for data elements used in the database. The database stores, metadata, that is, information about the database. These data elements are then used as part of a database, research project, or information system.

The data dictionary consists of two words, data, which represents data collected from several sources, and dictionary, which represents where this data is available. The data dictionary is an important part of the relational database because it provides additional information about the relationship between several tables in the database. A data dictionary in a DBMS helps users manage data in an orderly and orderly manner, thereby preventing data redundancy.

2. Weak Entity

Entity types that do not have key attributes of their own are called weak entity types. In contrast, regular entity types that do have a key attributes are sometimes called strong entity types.

Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attributes values. We call this other entity type the identifying or owner entity type and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type. A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity.

OR

2) a) Explain the Categories of Data Models.

Categories of Data Models

a) high-level or conceptual data models provide concepts that are close to the way many users perceive data.

Conceptual data model use concepts such as entities, attributes and relationship.

An entity represents a real-world object or concept, such as an employee or a project, that is described in the database.

M T W T F S S

COMPASS

Date :

An attribute represents some property of interest that further describes an entity, such as the employee's name or salary.

A relationship among two or more entities represents an interaction among the entities; for example, a works-on relationship between an employee and a project. Representational or implementation data models are the models used most frequently in traditional commercial DBMSs, and they include the widely-used relational data model.

b) Low-level or physical data models provides concepts that describe the details of how data is stored in the computer.

c) Representational (or implementational) data models, which provides concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer.
Representational data models represents data by using record structures and hence are sometimes called record-based data models.

Representational or implementation data models are the models used most frequently in traditional Commercial DBMSs, and they include the widely used relational data model. As well as in Legacy data models - the network and hierarchical models.

M T W T F S S

Date :

6) Explain the component modules of DBMS & their interactions with diagram.

A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, Consider the names, telephone numbers, and address of the people you know. This is a collection of related data with an implicit meaning and hence is a database.

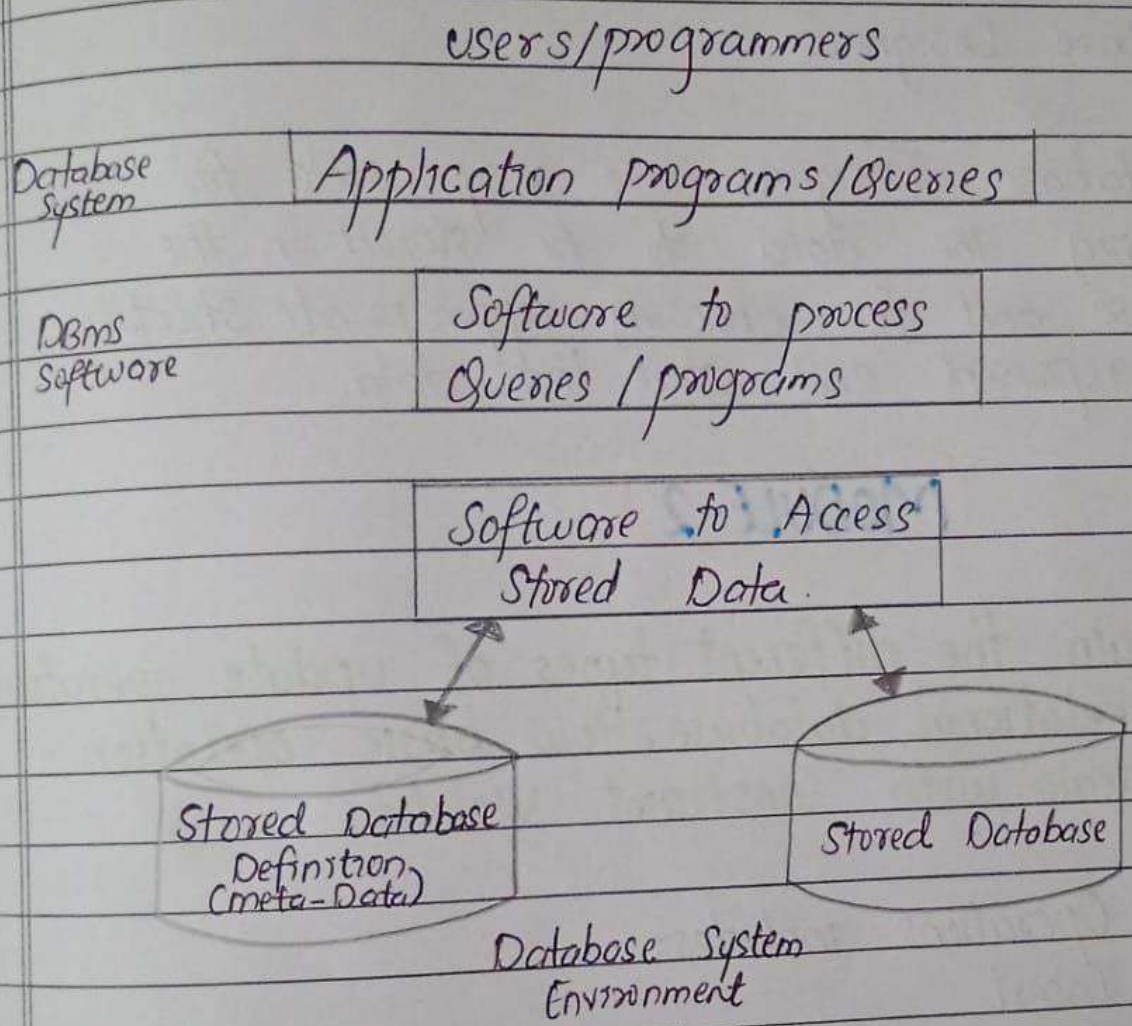
A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing and manipulating databases for various applications.

- Defining a database involves specifying the data types, structures and constraints for the data to be stored in the database.
- Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.
- Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- Sharing a database allows multiple users and programs to access the database simultaneously

Date :

- Application programs access the database by sending queries or request for data to the DBMS. A query causes some data to be retrieved.

Other important function provided by DBMS include protecting and maintaining it over long



Q) What are the responsibilities of DBA & database designers?

Database Administrators

In a database environment, the primary resource is the database itself and the secondary resource is the DBMS and related software. Administering these

resources as the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resource as needed.

Database Designers

Database Designers are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store this data.

MODULE-2

2) a) Explain the different types of update operations on relational database, How basic operation deals with Constraint Violation.

Update Operations includes:

1) Insert
2) Delete
3) Update
4) Transactions

Insert:

- Domain Constraint Violation: Some attributes value is not of correct domain Domain constraint gets violated only when a given value to the attributes

does not appear in the corresponding domain or in case it is not of the appropriate datatype.

- Entity integrity violation: Key of new tuple is null On inserting NULL values to any part of the primary key of a new tuple in the relation can cause violation of the entity integrity constraint.
- Key constraint violation: Key of new tuples is same as existing one. On inserting a value in the new tuple of a relation which is already existing in another tuple of the same relation, can cause violation of the key constraints.
- Referential integrity violation: On inserting a value in the foreign key of relation 1, for which there is no corresponding value in the primary key which is referenced to in relation 2, in such case Referential integrity is violated.

Delete:

On deleting the tuples in the relation, it may cause only violation of Referential integrity constraints.

Referential Integrity Constraints:

It causes violation only if the tuple in relation 1 is deleted which is referenced by foreign key from other tuples of table 2 in the database, if such deletion take place then the values in the tuple of the foreign key in table 2 will become empty. which will eventually violate Referential Integrity constraint.

Solutions that are possible to correct the violation to the referential integrity due to deletion are listed below:

1. Restrict
   Here we reject the deletion.

2. Cascade -
   Here if a record in the parent table is deleted, then the corresponding records in the child will automatically be deleted.

3. Set null or set default:
   Here we modify the referencing attributes values that cause violation and we either set NULL or change to another valid value.

3) Update:
- Key Constraint Violation: primary key is changed so as to became same as another tuple's.
- Referential integrity violation:
  → foreign key is changed and new one refers to nonexistent tuple
  → primary key is changed and now other tuple that had referred to this one violate the Constraint.

4) Transactions: This concept is relevant in the context where multiple users and/or application programs are accessing and updating the database concurrently. A transaction is a logical unit of work that may involve several accesses and/or updates to the database

(such as what might be required to reserve several seats on an airplane flight). The point is that, even though several transactions might be processed concurrently, the end result must be as though the transactions were carried out sequentially.

Explain Unary relational operation with examples.

The Select operation

- The Select operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.
- The SELECT operation is visualized as a horizontal partition of the relation into two sets of tuples. Those tuples that satisfy the condition are selected, and those tuple that do not satisfy the condition are discarded.
- In general, the SELECT operation is denoted by $\sigma_{<selection\ condition>}(R)$ where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.

Example 1: To Select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000, we can individually specify each of these two conditions with a SELECT operation as follows:

$\sigma_{Dno = 4}(EMPLOYEE)$

$\sigma_{Salary > 30000}(EMPLOYEE)$

M T W T F S S

- The Boolean expression specified in <selection condition> is made up of a number of clauses of the form
  <attribute name><comparison op><constant values
  or
  <attribute name><comparison op><attribute name>

The project operation:

- The SELECT operation choose some of the rows from the table while discarding other rows. The PROJECT operation, on the other hand, selects certain columns from the table and discards the other columns.

Example: To list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$\pi_{Lname, Fname, Salary}(EMPLOYEE)$

Result of SELECT and PROJECT operations

$\pi_{Lname, Fname, Salary}(EMPLOYEE)$

$\pi_{Sex, Salary}(EMPLOYEE)$

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

M T W T F S S

COMPASS

Date :

Sequence of Operations and the RENAME operation

For example, To retrieve the first name, last name and Salary of all employee who work in department number 5, we must apply a SELECT and a project operation

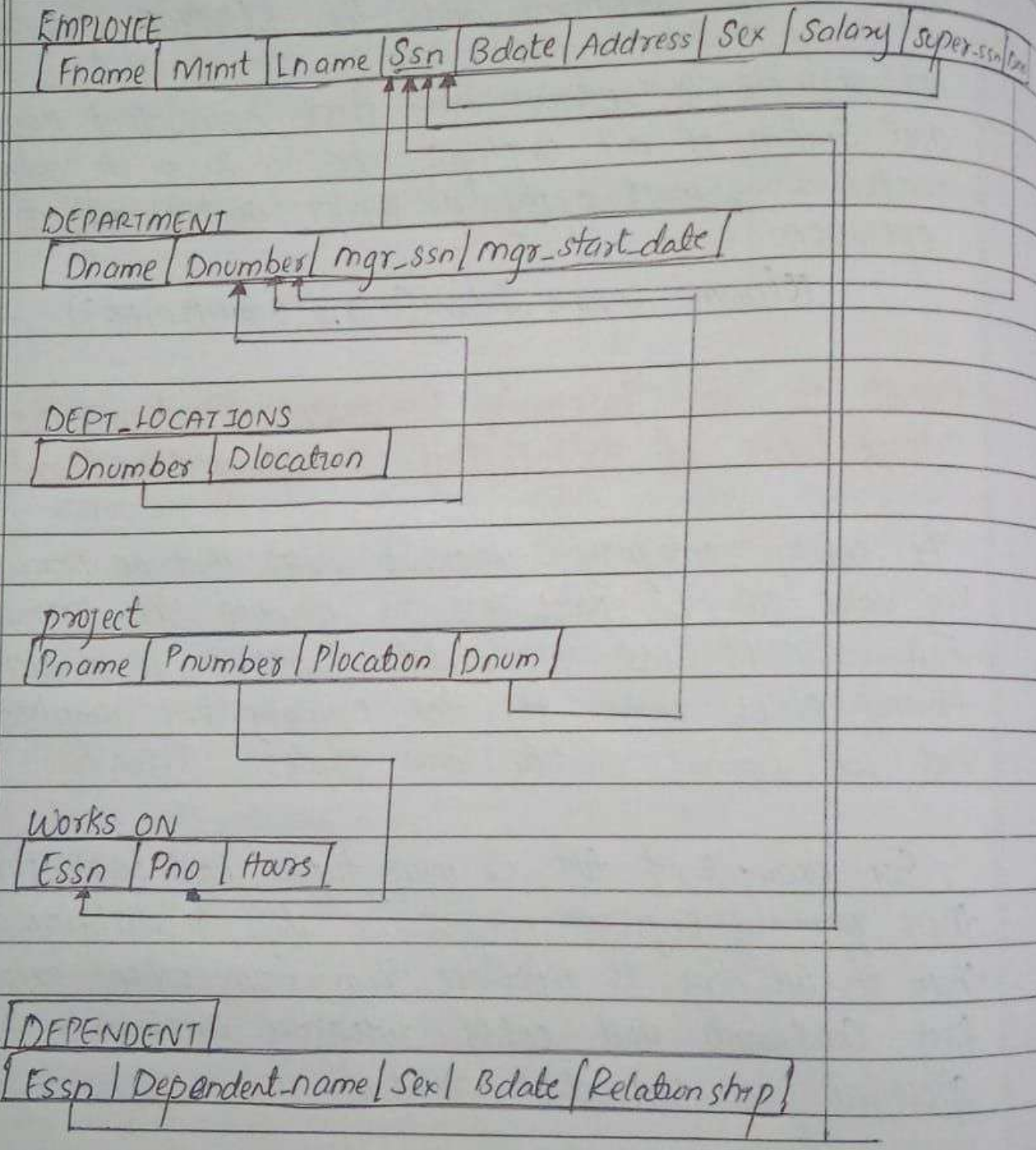
π(fname, Lname, Salary(σ Dno=5 (EMPLOYEE))

c) What is an Integrity Constraint? Explain the importance of Referential Integrity Constraint.

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

For example, if two or more tuples had NULL for their primary keys, we maynot be able to distinguish them if we try to reference them from other relation. Key Constraints and entity integrity constraints are specified on individual relation.

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.



OR

4 a) Explain the following relational algebra operation JOIN, DIFFERENCE, SELECT, UNION

JOIN OPERATION:

M T W T F S S

COMPASS

Date :

- The JOIN operation, denoted by ⋈, is used to combine related tuples from two relations into single-longer tuples
- The general form of a JOIN operation on two relation $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots B_n)$ is $R \underset{\text{join condition}}{\bowtie} S$.

To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the mgr_ssn value in the department tuple.

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{mgr\_ssn = ssn} EMPLOYEE$$

$$RESULT \leftarrow \pi_{Dname, Lname, Fname}(DEPT\_MGR)$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	..	Fname	Minit	Lname	Ssn	..
Research	5	333445555	..	Franklin	T	Wong	333445555	..
Administration	4	987654321	..	Jennifer	S	Wallace	987654321	..
Headquaters	1	888665555	..	James	E	Borg	888665555	..

Result of the JOIN operation $DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{mgr\_ssn = ssn} EMPLOYEE$

DIFFERENCE: The result of this operation, denoted by R-S, is a relation that includes all tuples that are in R but not in S.

- Example:

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johny	kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gillbert

Instructor

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

INSTRUCTOR - STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Slect - 2b

Union

The result of this Operation, denoted by RUS, is a relation that include all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated

Example:-

Date :

Student

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbala	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Fransic	Johnson
Ramesh	shah

Student ∪ Instructor

Fn	Ln
Susan	Yao
Ramesh	shah
Johnny	kohler
Barbala	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

b) Discuss the ER to Relational mapping algorithm with example for each step.

ER model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to relational schema, that is, it is possible to create

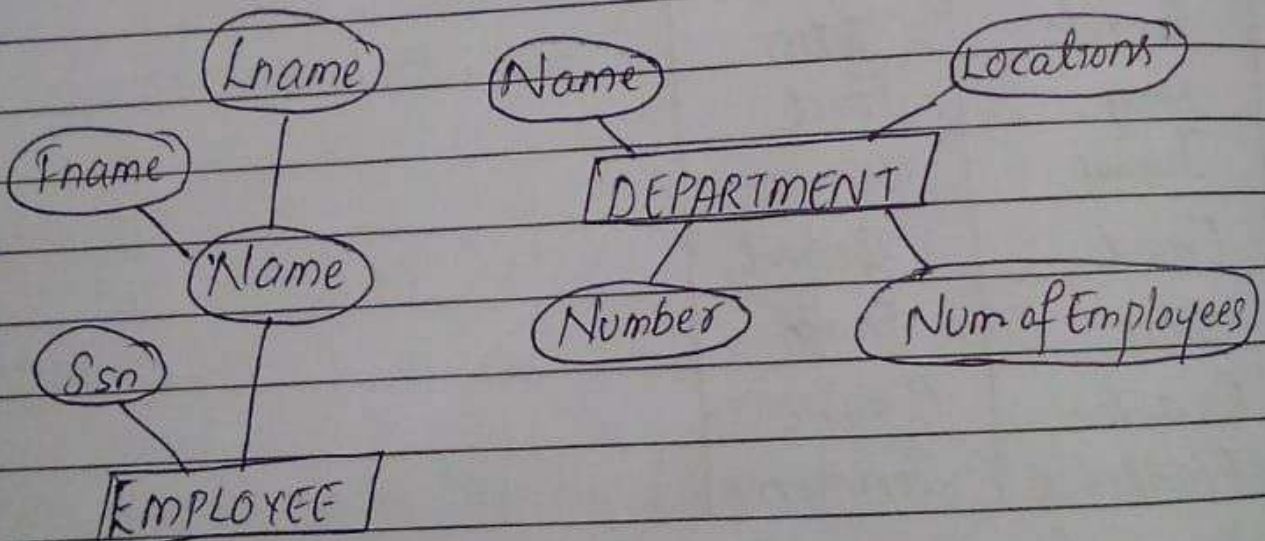
M T W T F S S

relational schema using ER diagram.

Step 1: For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.

Regular Entity Types

- For each regular/strong entity type, create a corresponding relation that includes all the simple attributes (including simple attributes of composite relations)
- Choose one of the key attributes as primary & if composite, the simple attributes together form the primary key.



EMPLOYEE

SSN	Lname	Fname

DEPARTMENT

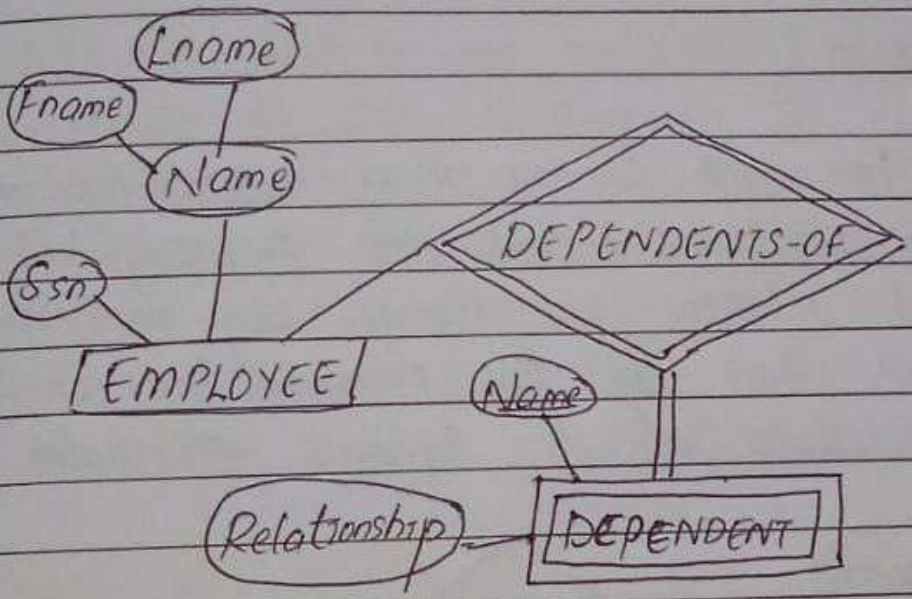
NUMBER	NAME

M T W T F S S

COMPASS

Date :

Step 2: Weak Entity Type :- For each weak entity type, create a corresponding relation that includes all the simple attributes. :- Add as a foreign key all of the primary key attributes(s) in the entity corresponding to the owner entity type. ::: The primary key is the combination of all the primary key attributes from the owner and the partial key of the weak entity, if any



DEPENDENT

EMPL-SSN	NAME	Relationship

Step 3: For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S.

M T W T F S S

DEPARTMENT

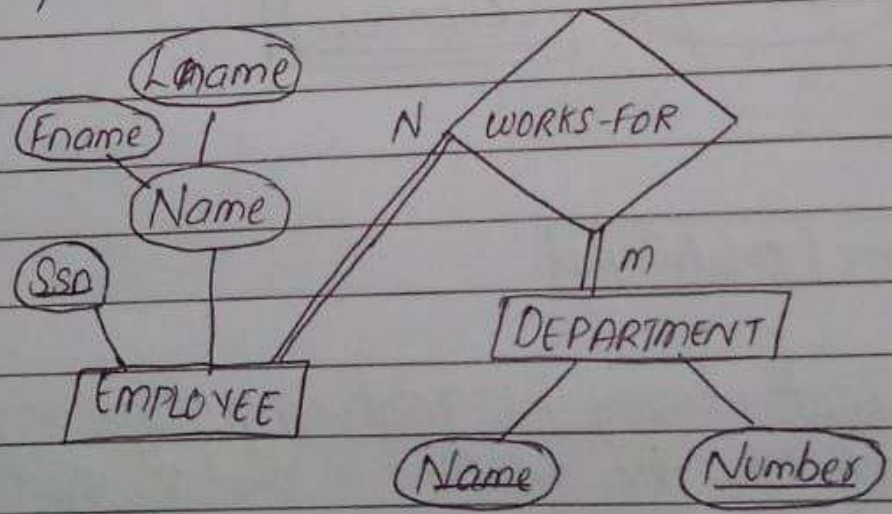
MANAGER-SSN	Start Date

Step 4: For each regular binary 1:N relationship type R identifying the relation (N) relation S Include the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S.

EMPLOYEE

SupervisorSSN

Step 5:- For each binary M:N relationship type R, Create a relation S. Include the primary keys of participant relations as foreign key in S. Their combination will be the primary key for S Simple attributes of R become attributes of S.



WORKS-FOR

EmployeeSSN	DeptNumber

COMBASS

Date :

Step 6: For each multi-valued attribute A, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation (entity type or relationship type) as a foreign key in R. The primary key of R is the combination of A and K.

DEP-LOCATION

Location	DEP-NUMBER

Step 7: For each n-ary relationship type R, where n>2, create a new relation S to represent R. Include the primary keys of the relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint >1.

6) Explain the relational algebra operation for set theory with example.

Binary operations from mathematical set theory:

1. UNION R1 ∪ R2
2. INTERSECTION R1 ∩ R2
3. SET DIFFERENCE R1 - R2
4. CARTESIAN PRODUCT : R1 × R2

The Union, Intersection, and Minus operations.

M T W T F S S

Date :

For example: To retrieve the Social Security number of all employees who either work in department 5 or directly supervise an employee who works in department 5

DEP5_EMPS ← σ Dno = 5 (EMPLOYEE)

RESULT1 ← π Ssn (DEP5_EMPS)

RESULT2 (Ssn) ← π Super_ssn (DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

- The relation RESULT1 has the SSn of all employee who work in department 5, where as RESULT2 has the Ssn of all employees who directly supervise an employee who works in department 5. The UNION operation produces the tuples that are in either RESULT1 or RESULT2 or both, while eliminating any duplicates Thus, the Ssn value '333445555' appears only once in the result.

RESULT1

Ssn
123456789
333445555
666884444
453453453

Result2

Ssn
333445555
888665555

Result

Ssn
123456789
333445555
666884444
453453453
888665555

RESULT ← RESULT1 ∪ RESULT2

M T W T F S S

COMPASS

Date :

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations R and S as follows:

- **UNION**: The result of this operation, denoted by R∪S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

- **INTERSECTION**: The result of this operation, denoted by R∩S, is a relation that includes all tuples that are in both R and S.

- **SET DIFFERENCE**: (or Minus): The result of this operation, denoted by R-S, is a relation that includes all tuples that are in R but not in S.

* The relations STUDENT and INSTRUCTOR are union compatible and their tuples represent the names of students and the names of instructors, respectively.

* The result of the UNION operation shows the names of all students and instructors. Note that duplicate tuples appear only one in the result.

* The result of INTERSECTION operation includes only those who are both student and instructors.

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wong
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT ∪ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wong
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT ∩ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah

STUDENT - INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wong
Ernest	Gilbert

INSTRUCTOR - STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

COMPASS
Date :

* UNION and INTERSECTION are commutative operation; that is, $R \cup S = S \cup R$ and $R \cap S = S \cap R$

* UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are also associative operations; that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

* The MINUS operation is not commutative; that is, in general, $R - S \neq S - R$

* INTERSECTION can be expressed in terms of union and set difference as follows.

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

* In SQL, there are three operation UNION, INTERSECT and EXCEPT that correspond to the set operations

CARTESIAN PRODUCT OPERATION

* The CARTESIAN PRODUCT operation - also known as CROSSPRODUCT or CROSS JOIN - which is denoted by X.

* Cartesian product produces a new element by combining every member (tuple) of one relation (set) with every member (tuple') from the other relation(s)

M T W T F S S

Date :

The result of R(A1, A2, .., An) X S(B1, B2 ... Bm) is a relation Q with degree n+m attributes Q(A1, A2, .. An, B1, B2, .., Bm).

The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations. We can SELECT related tuples from the two relations by specifying an appropriate selection condition after the cartesian product,

Example:-

STUDENT:

SINO	FName	LName
1	Albert	Singh
2	Nora	Fatehi

DETAILS:

ROLLNO	AGE
5	18
9	21

CROSS PRODUCT on STUDENT and DETAILS:

STUDENT X DETAILS

SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21
2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

M T W T F S S

COMBASS

Date :

MODULE-3

5) a) Illustrate insert, delete, update, alter & drop command in SQL.

a) Insert: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE NAME
(col 1, col 2, col 3, . . . col N)
VALUES (values 1, value 2, value 3, . . value N);
```

OR

```
INSERT INTO TABLE_NAME
values(value 1, valve 2, valve 3, . . . value N);
```

For example:

```
INSERT INTO EMP(EName, job) VALUES ("SCOTT", "MANAGER");
```

b) UPDATE: This command is used to update or modify the values of the column in the table

Syntax:

```
UPDATE table_name SET column 1 = value, column 2 = value,
ColumnN = value WHERE CONDITION;
```

For example:

```
UPDATE Emp SET Ename = 'SMITH' WHERE EmpNO = '1003';
```

c) DELETE: It is used to remove one or more row from a table

Syntax 1:

```
DELETE FROM table_name;
```

Syntax 2:

```
DELETE FROM table_name WHERE Condition;
```

Example: Delete all rows from emp table

```
DELETE From Emp
```

d) ALTER:

This statement/command can be used to delete modify or add constraints or column in an existing table

The 'ALTER TABLE' Statement

This statement is used to add, delete, modify or ~~add constraints~~ or column in an existing table

The 'ALTER TABLE' Statement with ADD/DROP COLUMN

You can use the ALTER TABLE Statement with ADD/DROP column command according to your need. If you wish to add a column, then you will use the ADD command, and if you wish to delete a column, then you will use the DROP column command.

COMPASS

Date :

Syntax

- ALTER TABLE TableName ADD ColumnName Datatype;
- ALTER TABLE TableName DROP COLUMN ColumnName;

Example:

-- ADD Column MobNO:
ALTER TABLE Emp ADD MobNo Number(10);
-- DROP Column MobNO:
ALTER TABLE Emp DROP COLUMN MobNO;

The 'ALTER TABLE' statement with ALTER/MODIFY COLUMN

The statement is used to change the datatype of an existing column in a table.

Syntax:

ALTER TABLE TableName ADD COLUMN ColumnName Datatype;

Example:
-- Add a column DOB and change the data type to Da
ALTER TABLE Emp ADD DOB date;

b) Explain informal design guidelines for relational Schema design

M T W T F S S

Date :

There are four informal guideline 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

1. Imparting clear Semantics to attributes in a relation

* When we group attributes to form a relation, the attributes must have a real world, meaning and proper interpretation associated with them.

* Consider the schema below:

EMPLOYEE

Ename	ssn	bdate	address	dnumber

* The easier to explain the semantics of the relation, the better the relation schema design will be.

EMPLOYEE-DETAILS

Ename	ssn	bdate	address	dnumber	Dpjct_nos

Date :

Guideline 1:

Design a relational schema so that it is easy to explain its meaning. Do not Combine attributes from multiple entity types and relationship types in to a single relation

* Violating guideline 1:
EMPLOYEE-DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr-ssn

* This combines the attributes the two real world entities EMPLOYEE and DEPARTMENT, which violates guideline 1.

Reducing the redundant information in tuples.

* One goal of Schema design is to minimize the storage space used by the base relations (and hence the corresponding files)

* 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 insertion anomalies deletion anomalies and modification anomalies.

COMPASS

M T W T F S S

Date :

3. NULL Values in tuples

* Null values lead to problems with understanding the meaning of the attributes
* Also it causes confusion in COUNT and SUM operations
* Also if NULL value comes in comparison in SELECT or JOIN operations, the result will be unpredictable

Guideline 3:

As far as possible, avoid placing attributes in a relation whose values may frequently NULL. If NULLS are unavoidable make sure that they do not apply to majority of the tuples.

4. Generation of Spurious tuples.

EMP-PROJ	Ssn	Pnumber	Hours	Ename	Pname	Plocation

EMP-LOCS	Ename	Plocation

EMP-PROJ1	Ssn	Pnumber	Hours	Pname	Plocation

The relation EMP-PROJ is decomposed in to two relations EMP-LOCS and EMP-PROJ1, But if the two tables are joined by JOIN operation, it will generate some spurious tuples, so the joined table will not be same as EMP-PROJ

COMPASS

Date :

Guideline 4

Design relation schema so that they can be joined with equality conditions on attributes and they are appropriately related pairs in a way it guarentee that no suprious tuples will be generated.

c. What is Functional dependency? Explain the inference rules for functional dependency with proof.

Functional Dependency:

* A functional dependency is a constraint between two sets of attributes from the database
* It describes relationship between attributes in a relation.
* A functional dependency, denoted by $X \rightarrow Y$. between two sets of attributes X and Y in a Relation R specifies a constraints that ifor any two tuples t1 and t2

if $t1[X] = t2[X]$ then $t1[Y] = t2[Y]$.

i.e, any values in X attributes is same then the corresponding values in Y attribute must also be same.

SI	Name	Address
1	A	A1
2	B	A2
3	C	A3
3	C	A4
4	D	A5
5	E	A6

M T W T F S S

Date :

SI -> Name

- Name is functionally dependent on SI
- SI determines Name
- If X -> Y, if t1[X] = t2[X] the t1[Y] = t2[Y]

Example:-

-> SI -> Name - True

-> SI -> Address - False

-> Name -> Address - False

Roll No	Name	Marks	Dept	Course
1	Smith	78	CSE	C1
2	Ramesh	60	EE	C1
3	Smith	78	CSE	C2
4	Ramesh	60	EE	C3
5	John	80	IT	C3
6	Bevin	80	EC	C2

Here the following Functional Dependency Exist

- RollNo -> Name (Since all values in roll no. are unique)
- RollNO -> marks (Since all values in roll no. are unique)
- RollNo, Name -> Marks (Since RollNo Name Combination is not repeating.

Here the following Functional Dependency does not Exist

- Name -> RollNo (Name like Smith & Ramesh are repeating)
- Dept -> Course (Dept CSE & EE repeating)
- Course -> Dept (Course C1, C2, C3 are repeating)

M T W T F S S

COMPASS

Date :

Types in Functional Dependency

1. Trival Functional Dependency
2. Non Trival

Inference Rules for Functional Dependencies

Definition:- An FD $X \rightarrow Y$ is inferred from or implied by a set of dependencies F specified on R if $X \rightarrow Y$ holds in every legal relation state r of R, that is, whenever r satisfies all the dependencies in F, $X \rightarrow Y$ also holds in r.

Proof of IR1:

Suppose that $X \supseteq Y$ and two tuples t1 and t2 exist in some relation instance r of R such that $t1[X] = t2[X]$

Then $t1[Y] = t2[Y]$ because $X \supseteq Y$; hence, $X \rightarrow Y$ must hold in r.

IR2 (Augmentation rule):

$\{X \rightarrow Z\} \models XZ \rightarrow YZ$

Proof by IR2 (by contradiction):

Assume that $X \rightarrow Y$ holds in a relation instence r of R but that $XZ \rightarrow YZ$ does not hold. Then there must exist two tuples t1 and t2 in r such that

1) $t1[X] = t2[X]$, 2) $t1[Y] = t2[Y]$,

3) $t1[XZ] = t2[XZ]$, and 4) $t1[YZ] \neq t2[YZ]$.

M T W T F S S

Date :

This is not possible because from (1) & (3) we deduce (5) $t1[Z] = t2[Z]$, and from (2) & (5) we reduce (6) $t1[YZ] = t2[Y]$, contradicting (4).

IR3 (Transitive rule):

$\{X \to Y, Y \to Z\} = X \to Z$

Proof of IR3:

Assume that (1) $X \to Y$ and (2) $Y \to Z$ both hold in a relation r. Then for any two tuples t1 and t2 in r such that $t1[X] = t2[X]$, we must have (3) $t1[Y] = t2[Y]$, from assumption (1); hence we must also have (4) $t1[Z] = t2[Z]$ from (3) and assumption (2); thus $X \to Z$ must hold in r.

IR4 (decomposition, or projective, rule): $\{X \to YZ\} \models X \to Y$

Proof of IR4:

1) $X \to YZ$ (given)

2) $YZ \to Y$ (using IR, rule)

3) $X \to Y$ (using IR_3 on 1 and 2)

IR5 (Union, or additive, rule):

$\{X \to Y, X \to Z\} \models X \to YZ$

1. $X \to Y$ (given)

2. $X \to Z$ (given)

3. $X \to XY$ (using IR2 on 1 by augmenting with X; notice that $XX = X$)

M T W T F S S

COMPASS

Date :

4. XY -> YZ (using IR2 on 2 by augmenting with Y)
5. X -> YZ (using IR3 on 3 and 4)

IR6 (pseudotransitive rule):
{X -> Y, WY -> Z} |= WX -> Z.

1. X -> Y (given)
2. WY -> Z (given)
3. WX -> WY (using IR_2 on 1 by augmenting with W)
4. WX -> Z (using IR_3 on 3 and 2)

6) b) Explain the types of update anomalies in SQL with an example.

There are 3 types of update anomalies
1) Insertion Anomaly
2) Update Anomaly
3) Deletion Anomaly

Insertion anomaly:

EMPLOYEE-DEPT	Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr-ssn

1) Here to enter the details of an employee, details of his department also must be entered. It results in repetition of details for each employee in the same department.
2) It is difficult to enter the details of a department with no employee, it results in null values for attributes

M T W T F S S

Date :

Deletion anomaly:

EMPLOYEE-DEPT	Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr-ssn

If the details of the last employee of a department is deleted, then the details of that department also no longer exist in the database.

Modification anomaly:
In the above schema, if the value one attribute (eg: mgrssn of a department) is changed that value should be updated for all the employee tuples in the same department.

EmpId	Ename	DeptName	Salary	Course_Title	Date_Completed
100	Ali	Marketing	4800	SPSS	6/19/2011
100	Ali	Marketing	4800	Surveys	10/7/2011
140	Said	Accounting	5200	Tax Acc	12/8/2011
110	ahmed	Infosystem	4300	Visual Basic	1/12/2011
110	ahmed	Infosystem	4300	C++	4/22/2011
150	karthi	Marketing	4200	SPSS	6/19/2011
150	karthi	Marketing	4200	Java	8/12/2011

Delete Anomaly:

Suppose delete empid 140, it also delete the Course_title and date_completed also

EmpId	Ename	DeptName	Salary	Course_Title	Date_Completed
140	Said	Accounting	5200	Tax Acc	12/8/2011

M T W T F S S

COMPASS

Date :

Insert Anomaly

Because, in the new record empid is available but course_title is NULL. As per entity integrity rule primary key must have not null value

EmpId	Ename	DeptName	Salary	Course_Title	Date_completed
100	Ala	Marketing	4800	NULL	6/19/2011

This is insert anomaly

Update Anomaly

Once record change the Salary as 4500 and another record missed for updating the changes is called updater anomaly

EmpId	Ename	DeptName	Salary	Course_Title	Date_Completed
150	Karthi	Marketing	4200	SPSS	6/19/2011
150	Karthi	Marketing	4500	Java	8/12/2011

This is update anomaly

COMPASS

Date :

Insert Anomaly

Because, in the new record empid is available but course_title is NULL. As per entity integrity rule primary key must have not null value

EmpId	Ename	DeptName	Salary	Course_Title	Date_completed
100	Ala	Marketing	4800	NULL	6/19/2011

This is insert anomaly

Update Anamoly

Once record change the Salary as 4500 and another record missed for updating the changes is called updater anomaly

EmpId	Ename	DeptName	Salary	Course_Title	Date_Completed
150	Karthi	Marketing	4200	SPSS	6/19/2011
150	Karthi	Marketing	4500	Java	8/12/2011

This is update anamaly

MODULE-4

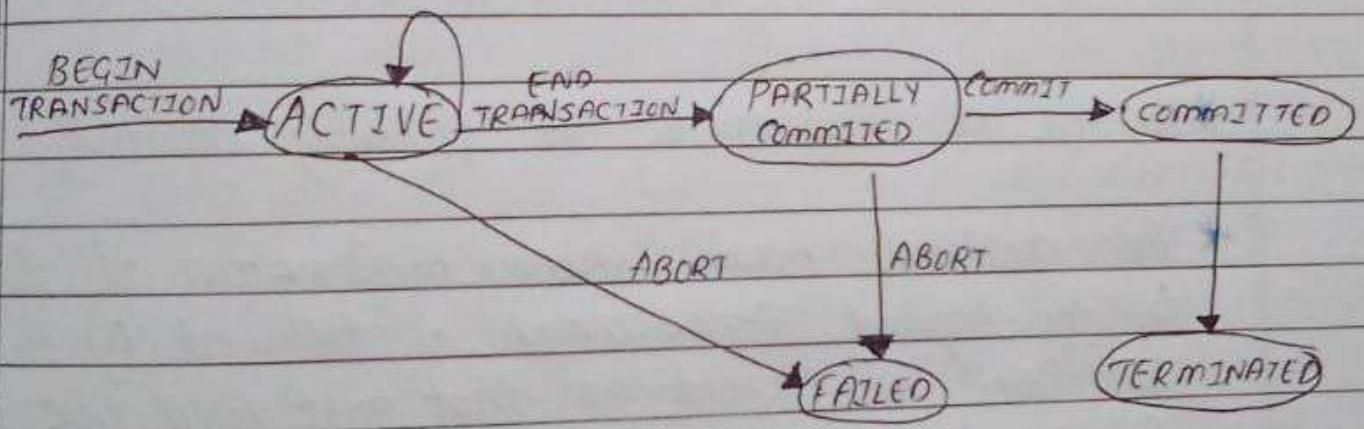
Demonstrate transaction states & additional operation.

Transaction States and Additional Operations.

A transaction is an atomic unit of work that is either completed in its entirely or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts, Hence the recovery manager keeps track of the following operation.

- BEGIN_TRANSACTION: This marks the beginning of transaction execution.
- READ_WRITE: These specify read or write operation on the database items that are executed as part of a transaction.
- END_TRANSACTION: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates Serializability or for some other reason.
- COMMIT_TRANSACTION: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- ROLLBACK (or ABORT): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Shows a state transaction diagram that describes how a transaction moves through its execution states. A transaction goes into an active state immediately after it start execution, where it can issue READ and WRITE operations when the transaction ends, it moves to the partially committed state. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log). Once this check is successful, the transaction is said to have reached its commit point and enters the committed state. Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.



b) Demonstrate working of Assertion & Triggers in database? Explain with an example.

* In SQL, users can specify general constraints - those that do not fall into any of the categories described so far via declarative assertions, using the CREATE ASSERTION statement of the DDL

M T W T F S S

Date :

* Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query

* For example, to specify the constraint that "the salary of an employee must not be greater than the salary of the manager of the department that the employee works

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK(NOT EXISTS
      (SELECT *
       from EMPLOYEE E, EMPLOYEE m, DEPARTMENT D
       WHERE E.SALARY > m.SALARY AND
             E.DNO = D.NUMBER AND
             D.MGRSSN = m.SSN));
```

* In SQL we can write the following assertion

* The constraint name SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a condition in parentheses that must hold true on every database state for the assertion to be satisfied.

* The constraint name can be used later to refer to the constraint or to modify or drop it.

* whenever some tuples in the database cause

M T W T F S S

COMPASS

Date :

the condition of an ASSERTION statement to evaluate to FALSE, the constraint is violated.

-> Another statement related to CREATE ASSERTION in SQL is CREATE TRIGGER, but trigger are used in a different way.

-> Trigger is used to specify the type of action to be taken when certain events occurs and when certain conditions are satisfied.

-> There are ~~typical~~ three Components in typical trigger

1> The event(s): These are usually update operations that are explicitly applied to the database. The person who writes the trigger must make sure that all possible events are accounted for. In some cases, it may be necessary to write more than one trigger to cover all possible cases. these events are specified after the keyword BEFORE, which means that the trigger should be executed before triggering operation is executed. An alternative is to use the keyword AFTER, which specifies that the trigger should be executed after the operation specified in the event is completed.

2> The condition that determines whether the role action should be executed: Once the triggering event has occurred, an optional Condition may be evaluted.

M T W T F S S

Date :

If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed. The condition is specified in the WHEN clause of the a condition is specified, it is first evaluated and only if it evaluates to true will the rule action be executed. The condition is specified in the WHEN clause of the trigger

3) The action is to be taken: The action is usually a sequence of SQL statement, but it could also be a database transaction or an external program that will be automatically executed.

* Rather than offering users only the option of aborting an operation (using ASSERTION) that causes a violation, the DBMS should make the following option available.

* It may be useful to specify a condition that if violated, causes some user to

* Example: A manager may want to be informed if an employee's travel expenses exceed a certain limit by receiving a message whenever this occurs.

* The action that the DBMS must take in this case is to send an appropriate message to the user. The condition is thus used to monitor the database.

* Other actions may be specified, such as executing a specific stored procedure or triggering other updates.

Trigger to insert a row in DEPT_LOCATIONS when a row is added to DEPARTMENT

```
Create trigger t after insert on DEPARTMENT
for each row
begin
    insert into DEPT_LOCATIONS values (G, 'TEXES');
end;

insert into DEPARTMENT values ("Accounts", 6,
453459827, '1995-04-23');
```

OR

8 a. Demonstrate the System Log in database transaction.

THE SYSTEM LOG

→ To recover from failure the system maintains a log to keep track of all transaction operations

M T W T F S S

Date :

-> The log is a file that is kept on disk
-> The following are the types of entries - called log records - that are written to the log file

-> [start_transaction, T], Indicates that transaction T has started execution.
-> [write_item, T, X, old_value, new_value]. Indicates that transaction T has changed the value of database item X from old-value to new_value.
-> [read_item, T, X]. Indicates that transaction T has read the value of database item X.
-> [commit, T]. Indicates that transaction T has completed successfully - and affirms that its effect can be committed, (recorded permanently) to the database.
-> [abort, T]. Indicates that transaction T has been aborted

* Occurs when all operations that access the database have completed successfully - and effect of operations recorded in the log

* Transaction writes a commit record into the log - If system failure occurs, can search for transactions with recorded start_transtion but no commit record.

* Force-writing the log buffer to disk
writing log buffer to disk before transaction reaches commit point

M T W T F S S

COMPASS

Date :

6) Discuss the ACID properties of database transaction.

=> Transactions should possess several properties, often called the ACID

Atomicity

- A transaction is an atomic unit of processing
- either the entire transaction takes place at once or doesn't happen at all.

Consistency preservation

- If transaction is completely executed from beginning to end without interference from other transactions.
- It should take the database from one consistent state to another.

Isolation.

- It states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- Existence of a transaction must not interfere other transaction.

Durability or permanency:-

- The changes applied to the database by a Committed transaction must persist in the database
- these changes must not be lost because of any failure.

M T W T F S S

Date :

Q) Explain stored procedure language in SQL with an example.

* It is sometimes useful to create database program modules (procedures or functions) that are stored and executed by the DBMS at the database server. These are historically known as database stored procedures, although they can be functions or procedures.

* Stored procedures are useful in the following circumstances:

a) If a database program is needed by several applications, it can be stored at the server and invoked by any of the application program. This reduces duplication of effort and improves software modularity.

b) Executing a program at the server can reduce data transfer and hence communication cost between the client and server in certain situations.

c) These procedures can enhance the modeling power provided by views by allowing more complex types of derived data to be made available to the database users.

```
SELECT C.cid, C.cname, COUNT(*) from Customer C,
Orders a WHERE C.cid = O.cid GROUP BY C.cid, C.cname
```

M T W T F S S

COMPASS

Date :

MODULE-5

Q: Explain the Two phase locking protocol used for Concurrency Control.

Two-phase Locking Techniques for Concurrency Control:

The main techniques used to control Concurrent execution of transactions are based on the concept of locking data items. A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one Lock for each data item in the database. Locks are used as a means of synchronizing the access by Concurrent transaction to the database items.

Types of Locks and System Lock Tables:

Locks are two kinds -

* Binary locks
* Shared/exclusive.

Binary Locks: A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1. Item X cannot be accessed by a database operation that requests the items If the value of the lock on X is 0, the item can be accessed

M T W T F S S

Date :

"STUDY FOR 25-MINUTES, THEN TAKE A 5-MINUTES BREAK"

we requested, and the lock value is changed to 1. we refer to the current value (or state) of the lock associated with item x as lock(x).

* Two operations, lock_item and unlock_item, are used with binary locking. A transaction requests assess to an item x first issuing a lock_item(x) operation.

When we use the binary locking scheme, the system must enforce the following rules:

1. A transaction T must issue the operation lock_item(x) before any read_item(x) or write_item(x) operations are performed in T.

2. A transaction T must issue the operation unlock_item(x) after all read_item(x) and write_item(x) operations are completed in T.

3. A transaction T will not issue a lock_item(x) operation if it already holds the lock on item X.

4. A transaction T will not issue an unlock_item(x) operation unless it already holds the lock on item X.

These rules can be enforced by the lock manager module of the DBMS.

Shared/Exclusive Locks:-

This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read Locks are shared because no data value is being changed.

When we use the shared / exclusive locking schema, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock(x) or write_lock(x) before any read_item(x) operation is performed in T.

2. A transaction T must issue the operation write_lak(x) before any write-item(x) operation is performed in T.

3. A transaction T must issue the operation unlock(x) after all read_item(x) and write_item(x) operations are Completed in T

4. A transaction T will not issue a read_lock(x) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.

5. A transaction T will not issue a write_lock(x) operation if it already holds a read(shared) lock or write (exclusive) lock on item X.

6. A transaction T will not issue an unlock(x) operation unless holds a read(shared) lock or a write (exclusive) lock on item X.

b) Define Schedule? Illustrate with an example.

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.

1. Serial Schedule:

Definition: Transaction are executed one after the other without any interleaving

Example:-

- Transaction T1: Read(A), write(A)
- Transaction T2: Read(B), write(B)
- Serial Schedule : T1 → T2 or T2 → T1

2. Non-Serial Schedule:

Definition:- Transaction are interleaved, meaning operation from different transaction are mixed.

M T W T F S S

COMPASS

Date :

Example:-

- Transaction T1 : Read(A), write(A)
- Transaction T2 : Read(B), write(B)
- Non-Serial Schedule : Read(A)(T1), Read(B)(T2), write(A)(T1), write(B)(T2)

3) Serializable Schedule:

Definition:- A non-serial schedule that ensures the same result as a serial schedule.

Types:-

1) Confict Serializable
2) View Serializable

Confict Serializable: Can be transformed into a serial schedule by swapping non-conficting operations.

View Serializable:- produce the same result as a serial schedule without overlapping transactions.

Example:-

- Transaction T1 : Read(A), write(A)
- Transaction T2 : Read(B), write(B)
- Serializable Schedule : Read(A)(T1), write(A)(T1), Read(B)(T2), write(B)(T2).

c) Why concurrency control is needed? Demonstrate with an example.

Purpose of Concurrency Control

- To ensure that the Isolation property is maintained while allowing transactions to execute concurrently (outcome of concurrent transactions should appear as though they were executed in isolation).
- To preserve database consistency by ensuring that the schedules of executing transactions are serializable.
- To resolve read-write and write-write conflicts among transactions

Concurrency Control Protocols (CCPs)

- A CCP is a set of rules enforced by the DBMS to ensure serializable schedules
- Also known as CCMs (Concurrency Control Methods)
- Main protocal known as 2PL (2-phase locking), which is based on locking the data items
- Other protocals use different techniques
- We first cover 2PL in some details, then give an overview of other techniques.

2PL Concurrency Control Protocal

Based on each transaction securing a lock on a data item before using it - Locking enfores mutual exclusion when accessing a data item - simplest kind of lock is a binary lock, which secures permission to Read or write a data item for a transaction

Example:- If T1 requests Lock(x) operation; the system grants the lock unless item X is already locked by another transaction. If requests is granted, data item X is locked on behalf of the requesting transaction T1, Unlocking operation removes the lock.

Example:- If T1 issue Unlock(x), data item X is mode available to all other transactions System maintains lock table to keep track of which items are locked by which transactions Lock(x) and Unlock(x) are hence system cells Transaction that requests a lock but do not have it granted can be placed on a waiting queque for the item Transaction T must unlock any items it had locked before T terminates.

OR

10) a) What is NOSQL? Explain the CAP theorem.

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. Unlike traditional relational database that use tables with predefined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.

M T W T F S S

Date:

CAP THEOREM

- The CAP Theorem states that it is not possible to guarantee all three of the desirable properties- Consistency, availability, and partion tolerance - at the same time in a distributed system with data replication.
- If this is the case then the distributed system designer would have to choose two properties out of the three guarentee.
- It is generally assumed that in many traditional (SQL) applications, guarenteeing Consistency through the ACID properties is important.
- On the other hand, in a NOSQL distributed data store, a weaker consistency level is often acceptable, and guarenteeing the other two properties (availability, partition tolerance) is important. Hence,
- weaker consistency levels are often used in NOSQL system instead of guarenteeing serializability In particular, a form of consistency known as 'eventual consistency' is often adopted in NOSQL System.

b) what are document based NOSQL System? basic operations CRUD in MongoDB.

* Document-based or document-oriented NOSQL systems typically store data collections of similar documents.
* These types of system are also sometimes known as 'document stores'.

* The individual documents somewhat resemble complex objects or XML documents. but a major difference between document-based systems versus object and object-relational systems and XML is that there is no requirement to specify a schema - rather, the documents are specified as 'self-describing data.'

* There are many document-based NOSQL systems, including MongoDB and CouchDB, among many others.

MongoDB CRUD operations:-

* MongoDb has several CRUD operations, where CRUD stands for (Create, read, update, delete). Document can be create and inserted into their collections using the insert operation, whose format is:

```
db.<collection_name>.insert(<document(s)>)
```

* The parameters of the insert operation can include either a single document or an array of document, the delete operation is called remove, and the format is:

```
db.<collection_name>.remove(<condition>)
```

* The documents to be removed from the collection are specified by a Boolean condition on some of the fields in the collection documents.

M T W T F S S

Date :

* If is also possible to use the update operation to replace an existing document with another one but keep the same object Id.

For read queries, the main command is called find, and format is:

db.<collection_name>.find(<condition>)

General Boolean conditions can be specified as <condition>, and the documents in the collection that return true are selected for the query result.

c) What is NOSQL Graph database? Explain Neo4j

* Another category of NOSQL systems is known as graph databases or graph-oriented NOSQL systems.
* The data is represented as a graph, which is a collection of vertices (nodes) and edges. Both nodes and edges can be labeled to indicate the type of entities and relationships they represent, and it is generally possible to store data associated with both individual nodes and individual edges.

Neo4j

* Neo4j, which is used in many applications.
* Neo4j is an open source system, and it is implemented in Java.

M T W T F S S

COMPASS

Date :

* The data model is Neo4j organizes data using the concepts of nodes and relation-ships. Both nodes and relationship can have properties, which store the data items associated with nodes and relationships.

* Nodes can have labels; the nodes that have the same label are grouped into a collection that identifies a subset of the nodes in the database graph for querying purpose.

* A node can have zero, one, or several labels.

* In Conventional graph theory, nodes and relationships are generally called vertices and edges.

* The Neo4j graph data model somewhat resembles how data is represented in the ER and EER models, but with some notable difference.

* Comparing the Neo4j graph model with ER/EER concepts, nodes correspond to entities, node labels corresponds to entity types and subclasses, relation-ships correspond to relationship instances, relationship types Correspond to relationship types, and properties correspond to attributes.

* Relationship is directed in Neo4j, but not in ER/EER.

Model Question Paper- I

Module – 1

Q.1 a Explain the types of end users with examples.

1. Casual End Users

- **Description:** These users occasionally access the database but do not have detailed knowledge of its structure. They typically use query languages to retrieve information.
- **Example:** A manager who occasionally queries the database to generate a sales report.

2. Naive or Parametric End Users

- **Description:** These users frequently use the database but have a limited understanding of its complexities. They often interact with the system through pre-defined functions or transactions.
- **Example:** A bank teller who uses a banking application to enter customer transactions and retrieve account information.

3. Sophisticated End Users

- **Description:** These users have a good understanding of the database and its capabilities. They interact with the database to perform complex queries and analyses.
- **Example:** A data analyst who writes SQL queries to extract, manipulate, and analyze large datasets for business insights.

4. Stand-alone Users

- **Description:** These users maintain personal databases using tools like Microsoft Access or SQLite. They create and manage their databases independently.
- **Example:** A small business owner who uses Microsoft Access to manage customer information and sales records.

5. System Analysts and Application Programmers

- **Description:** These users design, implement, and maintain database applications. They write application programs that interact with the database and ensure it meets the needs of the organization.
- **Example:** A software developer who creates and maintains an inventory management system that interfaces with a company's database.

6. Database Administrators (DBAs)

- **Description:** These users are responsible for the overall management of the database. They ensure its integrity, security, and performance, and manage user access.
- **Example:** A DBA who manages database backup and recovery, configures security settings, and optimizes database performance.

Q1 b.What are the advantages of using DBMS? Explain.

Using a Database Management System (DBMS) has numerous advantages. Here are some of the key benefits:

1. Data Redundancy and Inconsistency Control:

- **Redundancy Minimization**: A DBMS reduces data redundancy by integrating all the data into a single database, reducing the need for duplicate data.
- **Inconsistency Reduction**: By centralizing data storage, a DBMS ensures that updates are consistent across the system, preventing data anomalies.

2. Data Sharing:

- **Multi-User Environment**: A DBMS allows multiple users to access and work with the data simultaneously.
- **Controlled Access**: Users can be granted different levels of access, ensuring data security and integrity.

3. Improved Data Security:

- **Access Controls**: DBMSs provide robust mechanisms for controlling user access and permissions.
- **Encryption**: Sensitive data can be encrypted to protect it from unauthorized access.

4. Data Integrity:

- **Constraints**: Integrity constraints such as primary keys, foreign keys, and unique constraints help maintain the accuracy and consistency of data.
- **Consistency Rules**: DBMSs enforce data consistency rules automatically, reducing the chances of errors.

5. Backup and Recovery:

- **Automatic Backup**: DBMSs often include tools for automatic data backup, ensuring that data can be recovered in case of a failure.
- **Recovery Mechanisms**: DBMSs provide recovery solutions to restore data after crashes, ensuring minimal data loss.

6. Data Independence:

- **Logical Data Independence**: Changes to the logical schema do not affect the application programs.
- **Physical Data Independence**: Changes to the physical storage of data do not affect the logical schema or application programs.

7. Efficient Data Access:

- **Query Optimization**: DBMSs use sophisticated algorithms to optimize query performance, making data retrieval efficient.
- **Indexing**: Indexes can be created to improve the speed of data retrieval operations.

8. Concurrent Access:

- **Concurrency Control**: DBMSs handle concurrent access to data, ensuring that multiple users can perform transactions simultaneously without conflicts.
- **Locking Mechanisms**: These prevent conflicts and ensure data consistency during concurrent operations.

9. Enhanced Data Integration:

- **Centralized Data Management**: Data from various sources can be integrated into a single database, providing a unified view of the data.
- **Data Warehousing**: DBMSs can support data warehousing for better data analysis and reporting.

10. Application Development Efficiency:

- **Standardized Query Language**: SQL provides a standardized language for querying and managing data, making it easier to develop and maintain applications.
- **DBMS Tools**: Many DBMSs offer tools and interfaces that simplify the development process.

11. Cost Efficiency:

- **Reduced Storage Costs**: By minimizing redundancy, storage costs are reduced.
- **Maintenance Savings**: Centralized data management reduces the cost and effort involved in maintaining data.

12. Improved Decision Making:

- **Data Analysis**: A DBMS can provide advanced data analysis tools, enabling better decision-making through data insights.
- **Reporting**: Various reporting tools integrated into DBMSs help generate insightful reports for decision-makers.

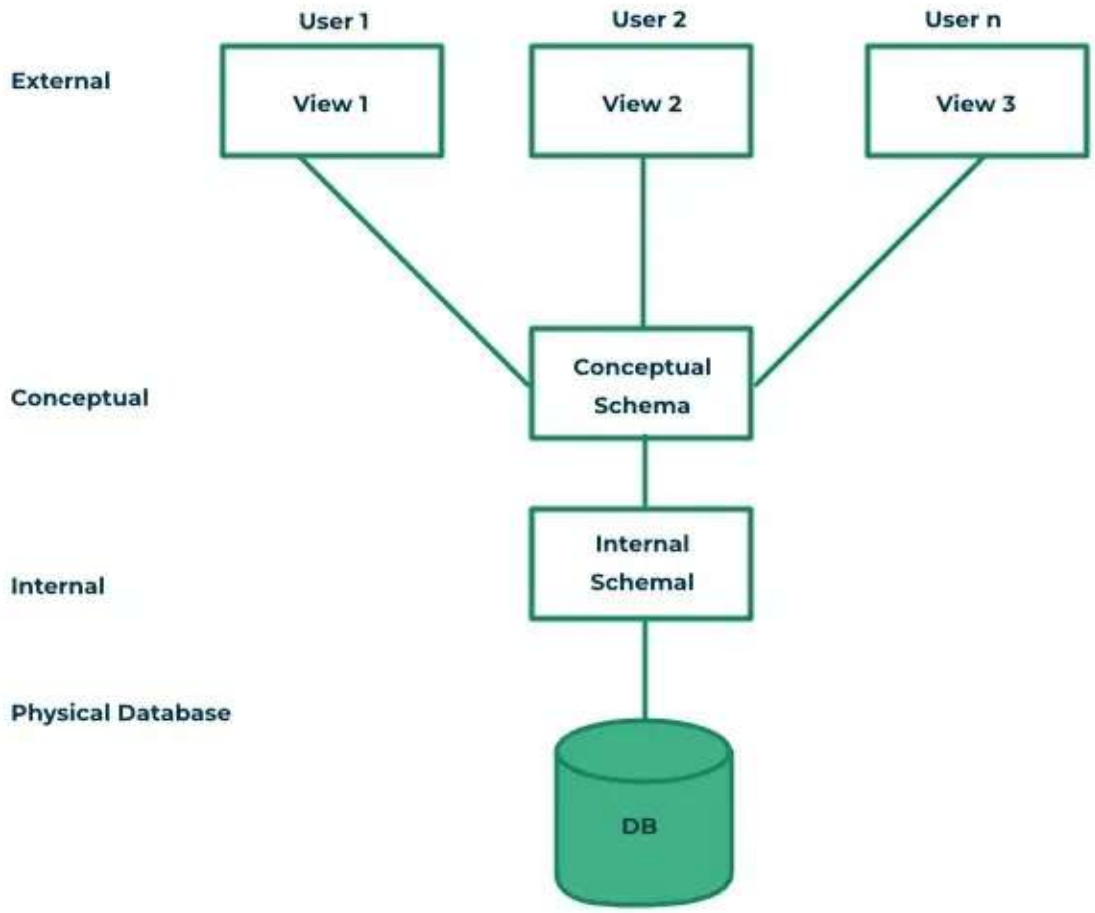
Q1.c Describe the characteristics of database.

Databases are fundamental components in modern computing and information management. Here are some key characteristics of databases:

1. **Structured Data Storage**: Databases organize data into tables or collections with predefined schemas. This structure helps in efficiently storing, retrieving, and managing data.
2. **Data Integrity**: Databases enforce rules and constraints to ensure the accuracy and consistency of the data. Common integrity constraints include primary keys (which uniquely identify records), foreign keys (which establish relationships between tables), and various data validation rules.
3. **Data Management**: Databases support various data manipulation operations such as creating, reading, updating, and deleting data (often abbreviated as CRUD operations). This functionality is facilitated through database management systems (DBMS) and query languages like SQL.
4. **Concurrency Control**: Databases manage concurrent access by multiple users or processes to ensure that transactions are processed reliably without conflicts. This involves mechanisms for locking data and managing transactions.
5. **Transaction Management**: Databases support transactions, which are sequences of operations that are executed as a single unit. Transactions are designed to ensure that operations are completed fully and correctly or not executed at all, maintaining data consistency (following the ACID properties: Atomicity, Consistency, Isolation, Durability).
6. **Scalability and Performance**: Databases are designed to handle large volumes of data and high transaction rates efficiently. They use indexing, optimization techniques, and caching to improve performance and scalability.
7. **Data Security**: Databases implement security measures to protect data from unauthorized access and breaches. This includes user authentication, authorization, encryption, and auditing.
8. **Backup and Recovery**: Databases provide mechanisms for backing up data and recovering it in case of failures or data loss. Regular backups and recovery plans are essential for data protection and continuity.
9. **Data Retrieval and Querying**: Databases offer powerful querying capabilities to retrieve and manipulate data. Query languages like SQL enable users to perform complex searches, aggregations, and data analysis.
10. **Data Relationships**: Databases can represent complex relationships between data entities through mechanisms such as foreign keys and joins. This allows for the efficient organization and retrieval of related information.
11. **Flexibility and Extensibility**: Many databases are designed to be flexible and can adapt to changes in data requirements. Some modern databases support schema evolution and dynamic changes.
12. **Data Redundancy Reduction**: Through normalization processes, databases reduce redundancy and improve data consistency by organizing data into related tables and minimizing duplication.

Q2.a Explain three schema architecture. Why mappings b/w schema levels are required?

The three-schema architecture is a framework used in database systems to manage data abstraction and separation. It involves three distinct levels of schemas:



1. **Internal Schema (Physical Schema)**:
   - **Definition**: This schema describes how data is physically stored in the database. It deals with the storage structure, file organization, indexing, and access methods.
   - **Purpose**: It provides the details about how data is organized and managed on the storage medium, ensuring efficient data retrieval and management.

- **Focus**: Implementation details, such as storage devices, data structures, and access paths.

2. **Conceptual Schema (Logical Schema)**:
   - **Definition**: This schema represents the entire database's logical structure. It defines the database's entities, relationships, and constraints without considering how they are physically stored.
   - **Purpose**: It provides a unified and coherent view of the data, ensuring consistency and integrity across the database. It serves as an abstraction layer between the physical storage and user views.
   - **Focus**: Data organization, relationships, and constraints without concern for physical storage details.
3. **External Schema (View Schema)**:
   - **Definition**: This schema defines different user views or perspectives of the database. Each view is a subset of the database that is tailored to specific user needs or application requirements.
   - **Purpose**: It provides customized views of the data for different users or applications, enhancing security and usability by restricting access to only the relevant parts of the database.
   - **Focus**: User interactions and data presentation tailored to specific needs.

Importance of Mappings Between Schema Levels

Mappings between these schema levels are essential for several reasons:

1. **Data Abstraction**:
   - **Purpose**: Mappings allow for data abstraction, separating the physical storage of data from its logical representation and user views. This separation ensures that changes to the physical storage (e.g., changing indexing methods or storage formats) do not impact the logical schema or user views.
2. **Flexibility and Independence**:
   - **Purpose**: By defining mappings between schemas, databases can evolve independently at different levels. For instance, if the internal schema changes (e.g., due to hardware upgrades), the conceptual schema and external schemas can remain unchanged, preserving the consistency of user views and logical data organization.
3. **Consistency and Integrity**:
   - **Purpose**: Mappings ensure that the logical schema accurately reflects the data stored at the physical level and that user views are consistently represented. This helps in maintaining data integrity across various schemas.
4. **User-Specific Views**:
   - **Purpose**: Mappings allow different users to interact with the database according to their needs without affecting the overall database structure. Users can work with customized views (external schemas) tailored to their specific roles or applications, while the underlying data structure (conceptual schema) remains consistent.
5. **Simplified Management**:

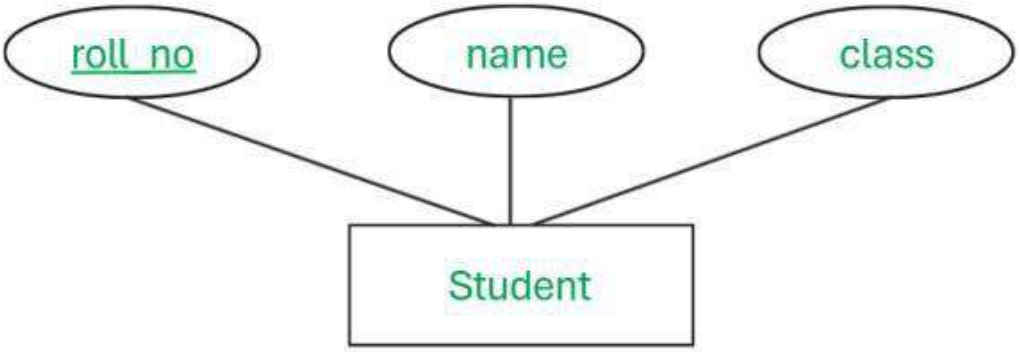
- **Purpose**: Managing and maintaining the database becomes easier when different schemas are mapped correctly. Database administrators can optimize physical storage and indexing without altering the logical schema or user views.

Q2.b Explain the different types of attributes in ER model.

1. Simple Attribute

An attribute that cannot be further subdivided into components is a simple attribute.

Example: The roll number of a student, the ID number of an employee, gender, and many more.

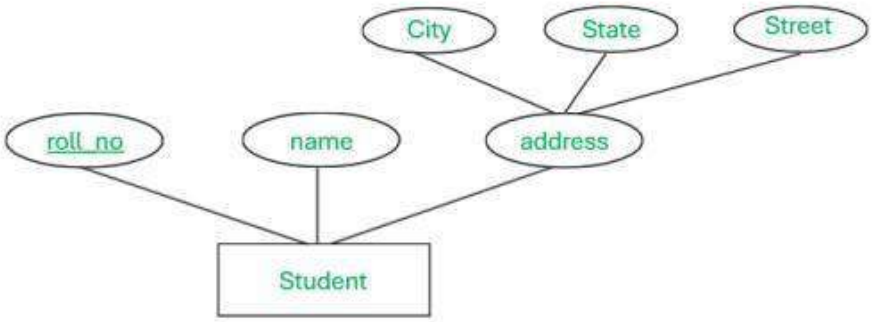


Simple Attribute

2.Composite Attribute

An attribute that can be split into components is a composite attribute.

Example: The address can be further split into house number, street number, city, state, country, and pin code, the name can also be split into first name middle name, and last name.

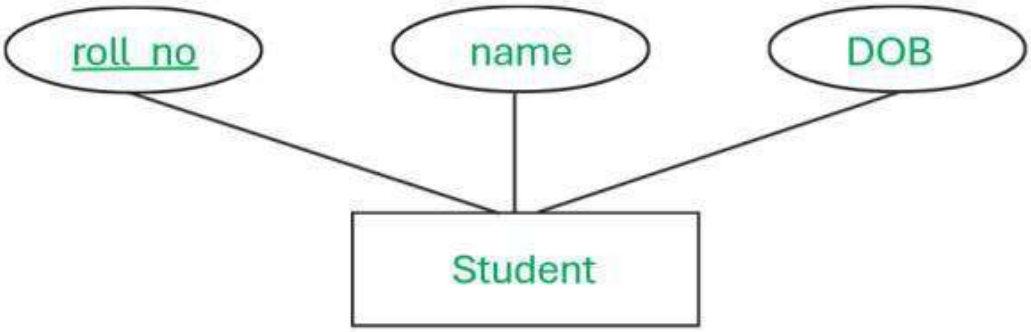


Composite Attribute

3. Single-Valued Attribute

The attribute which takes up only a single value for each entity instance is a single-valued attribute.

Example: The age of a student, Aadhar card number.

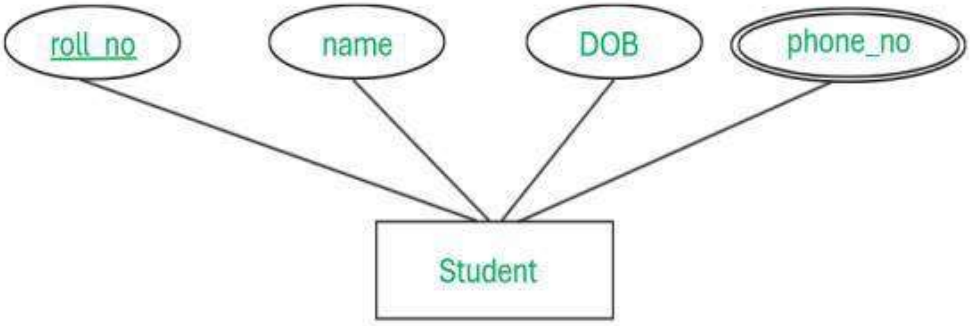


Single-Valued

4. Multi-Valued Attribute

The attribute which takes up more than a single value for each entity instance is a multi-valued attribute. And it is represented by double oval shape.

Example: Phone number of a student: Landline and mobile.

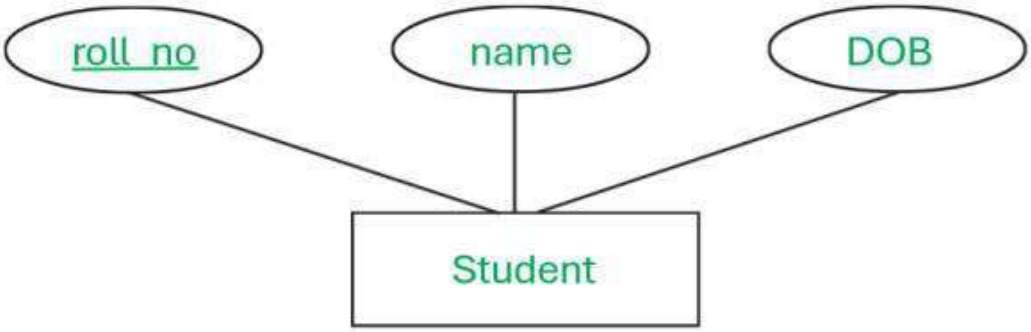


Multi-valued

5. Stored Attribute

The stored attribute are those attribute which doesn't require any type of further update since they are stored in the database.

Example: DOB(Date of birth) is the stored attribute.

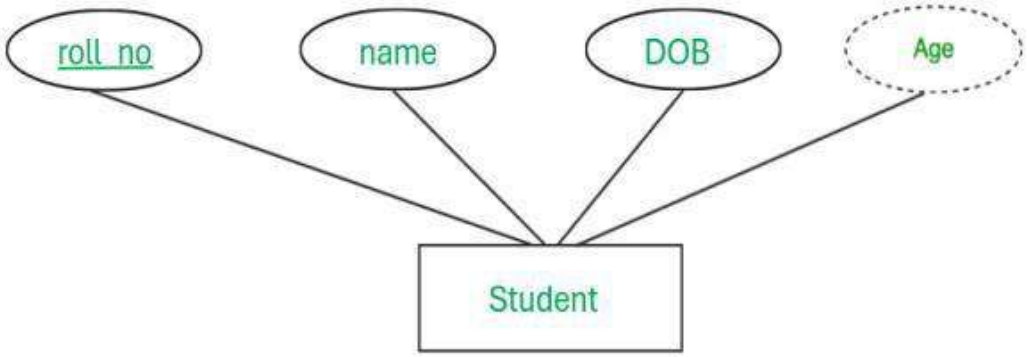


Stored-attribute

6. Derived Attribute

An attribute that can be derived from other attributes is derived attributes. And it is represented by dotted oval shape.

Example: Total and average marks of a student, age of an employee that is derived from date of birth.

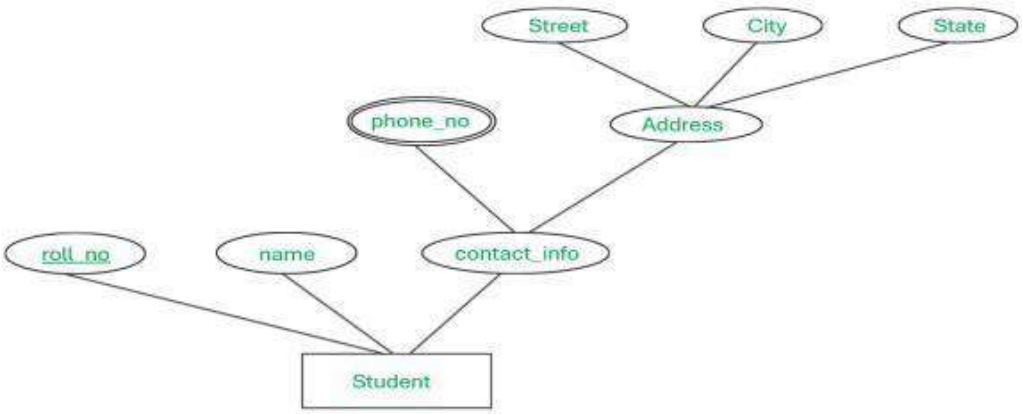


Derived-attribute

7. Complex Attribute

Those attributes, which can be formed by the nesting of composite and multi-valued attributes, are called "*Complex Attributes*". These attributes are rarely used in DBMS(DataBase Management System). That's why they are not so popular.

Example: Address because address contain composite value like street, city, state, PIN code and also multivalued because one people has more that one house address.



Complex-attribute

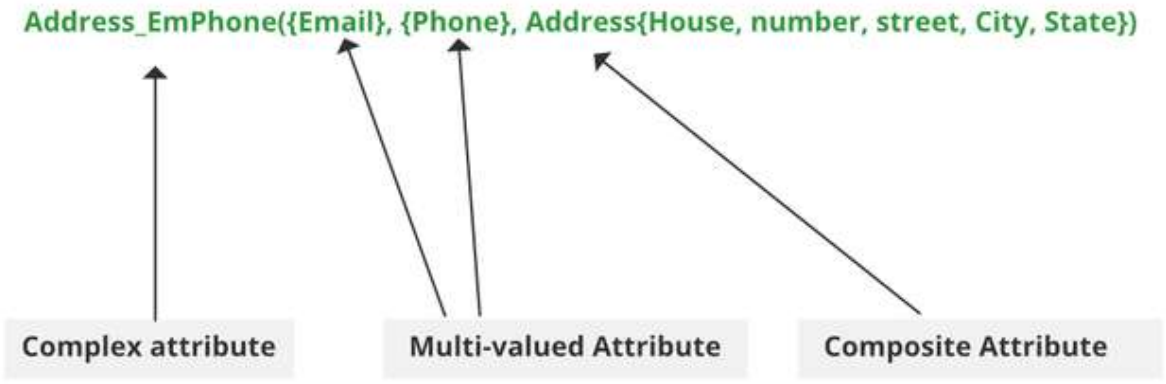
Representation

Complex attributes are the nesting of two or more composite and multi-valued attributes. Therefore, these multi-valued and composite attributes are called 'Components' of complex attributes.

These components are grouped between parentheses '( )' and multi-valued attributes between curly braces '{ }', Components are separated by commas ', '.

For example**:** let us consider a person having multiple phone numbers, emails, and an address.

Here, phone number and email are examples of multi-valued attributes and address is an example of the composite attribute, because it can be divided into house number, street, city, and state.



Complex attributes

Components

Email, Phone number, Address(All are separated by commas and multi-valued components are represented between curly braces).
Complex Attribute: Address_EmPhone(You can choose any name).

8. Key attribute

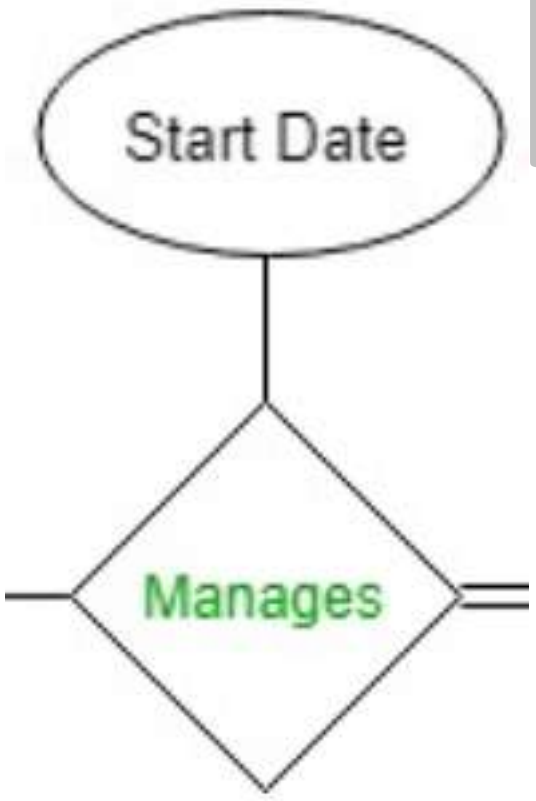
Key attributes are those attributes that can uniquely identify the entity in the entity set.
Example: Roll-No is the key attribute because it can uniquely identify the student.

9. Null Attribute

This attribute can take NULL value when entity does not have value for it.
Example –The 'Net Banking Active Bin' attribute gives weather particular customer having net banking facility activated or not activated.
For bank which does not offer facility of net banking in customer table 'Net Banking Active Bin' attribute is always null till Net banking facility is not activated as this attribute indicates Bank offers net banking facility or does not offers.

10. Descriptive Attribute

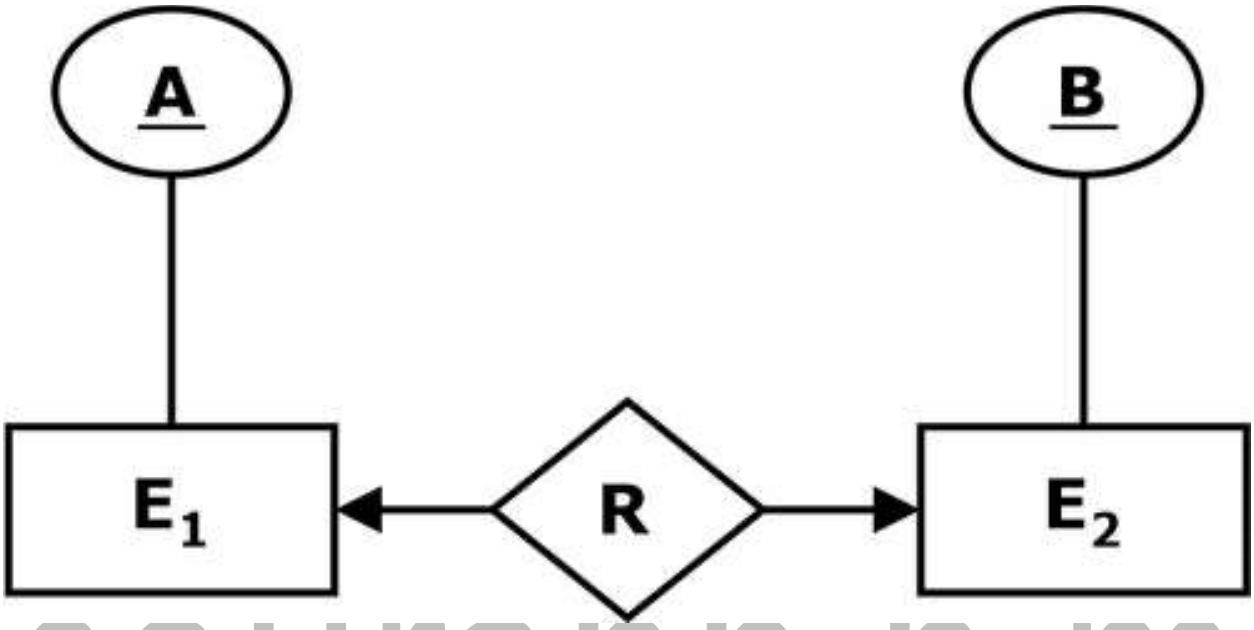
Descriptive attribute give information about the relationship set example given below. Here Start Date is the descriptive attribute of Manages relationship.



Descriptive-Attribute

Q2.cExplain the following.
1. Cardinality Ratio 2. Weal Entity

1. Cardinality Ratio



Definition: Cardinality ratio describes the number of instances of one entity that are associated with the number of instances of another entity in a relationship. It specifies how many instances of an entity participate in a relationship with instances of another entity.

Types of Cardinality Ratios:

1. **One-to-One (1:1)**:
   - **Definition**: An instance of entity A is associated with at most one instance of entity B, and vice versa.
   - **Example**: In a database of employees and their company-issued cars, each employee might be assigned exactly one car, and each car is assigned to exactly one employee.
2. **One-to-Many (1**

   ):

   - **Definition**: An instance of entity A can be associated with multiple instances of entity B, but an instance of entity B is associated with at most one instance of entity A.
   - **Example**: A single department (entity A) can have multiple employees (entity B), but each employee belongs to only one department.
3. **Many-to-One (N:1)**:

- **Definition**: Multiple instances of entity A can be associated with a single instance of entity B, but each instance of entity B is associated with at most one instance of entity A.
- **Example**: Many employees (entity A) can work under one manager (entity B), but each manager supervises only one department (entity B).

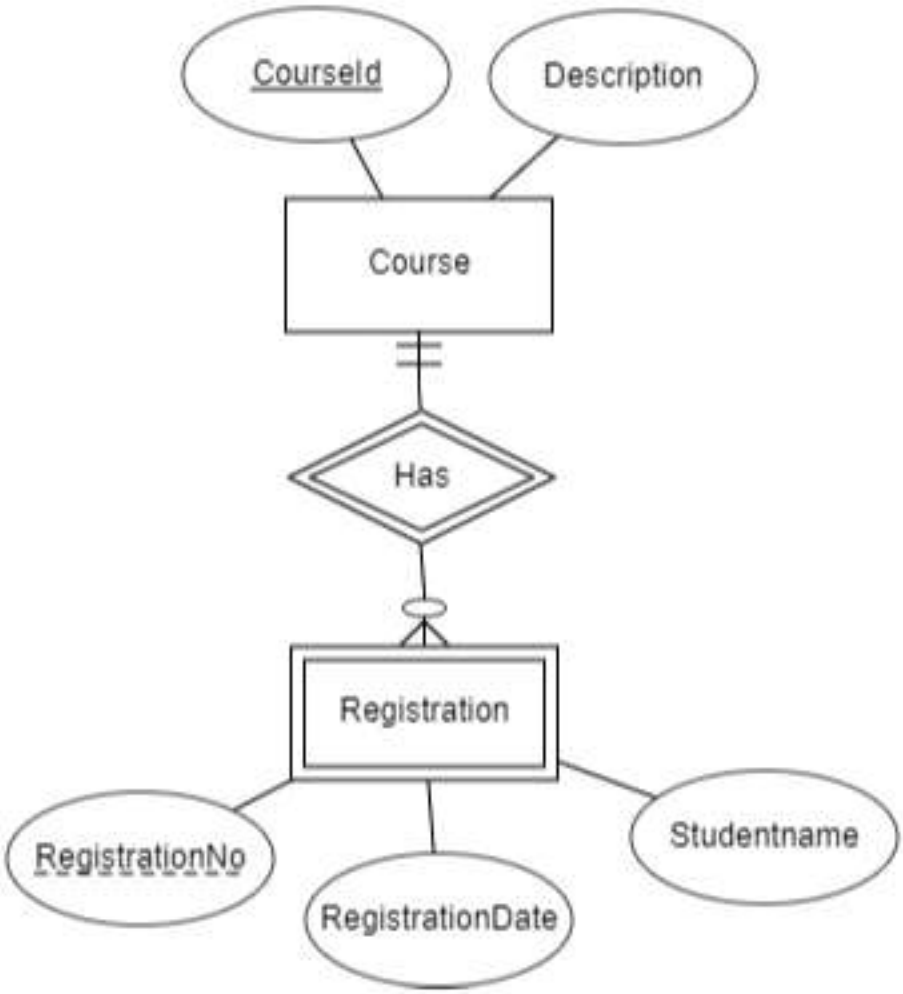
4. **Many-to-Many (M**

   ):

   - **Definition**: Multiple instances of entity A can be associated with multiple instances of entity B, and vice versa.
   - **Example**: Students (entity A) can enroll in multiple courses (entity B), and each course can have multiple students.

Importance: Cardinality ratios are crucial for understanding how entities relate to each other, which helps in designing the database schema and ensuring that relationships are represented correctly.

2. Weak Entity



Definition: A weak entity is an entity that cannot be uniquely identified by its own attributes alone. It relies on a "strong" or "owner" entity and a relationship with that entity to ensure its uniqueness.

Characteristics of Weak Entities:

1. **Lack of Key Attribute**: A weak entity does not have a primary key of its own. Instead, it has a partial key, which is an attribute or set of attributes that can uniquely identify the weak entity in conjunction with the primary key of the strong entity.
2. **Dependence on Strong Entity**: The weak entity is dependent on a strong entity, which provides part of the identifying information. This relationship is typically depicted by a double rectangle (for the weak entity) and a double diamond (for the identifying relationship) in ER diagrams.

3. **Existence Dependency**: A weak entity cannot exist without its associated strong entity. If the strong entity is deleted, the weak entity is also deleted.

Example: Consider an example involving an entity `Order` and a weak entity `OrderItem`. An `Order` entity can have multiple `OrderItem` entities. Here, `OrderItem` may not have a unique identifier on its own; it relies on the `Order` entity's primary key plus a partial key (e.g., `ItemNumber`) to identify each `OrderItem`.

ER Diagram Representation:

- **Weak Entity**: Represented by a double rectangle.
- **Identifying Relationship**: Represented by a double diamond.
- **Partial Key**: Underlined with a dashed line.

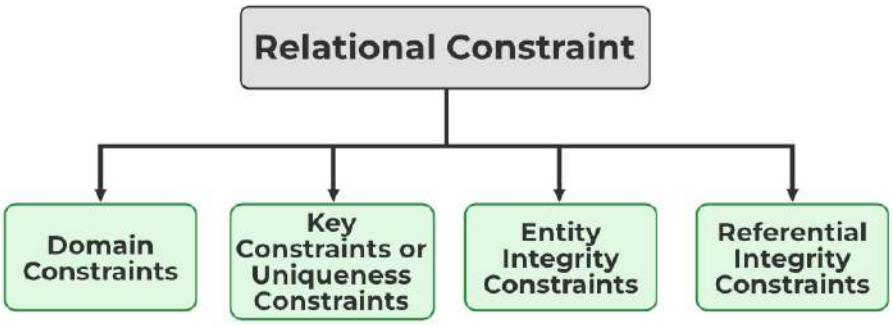
Importance: Weak entities are important for representing complex real-world scenarios where certain entities cannot be uniquely identified without referencing another entity. They help in modeling hierarchical or dependent relationships in a database schema.

Q3.a Explain the different Relational Model constraint

Relational Constraints

These are the restrictions or sets of rules imposed on the database contents. It validates the quality of the database. It validates the various operations like data insertion, updation, and other processes that have to be performed without affecting the integrity of the data. It protects us against threats/damages to the database. Mainly Constraints on the relational database are of 4 types

- Domain constraints
- Key constraints or Uniqueness Constraints
- Entity Integrity constraints
- Referential integrity constraints



1. Domain Constraints

- Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.
- We perform a datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we can't give it values other than int datatype.

Example:

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

Explanation: In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

2. Key Constraints or Uniqueness Constraints

- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as the primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
- Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

Example:

EID	Name	Phone
01	Bikash	6000000009
02	Paul	9000090009
01	Tuhin	9234567892

Explanation: In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

3.Entity Integrity Constraints

- Entity Integrity constraints say that no primary key can take a NULL value, since using the primary key we identify each tuple uniquely in a relation.

Example:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

Explanation: In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

4. Referential Integrity Constraints

- The Referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
- This constraint is enforced through a foreign key, when an attribute in the foreign key of relation R1 has the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
- The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

Example:

EID	Name	DNO
01	Divine	12
02	Dino	22
04	Vivian	14

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

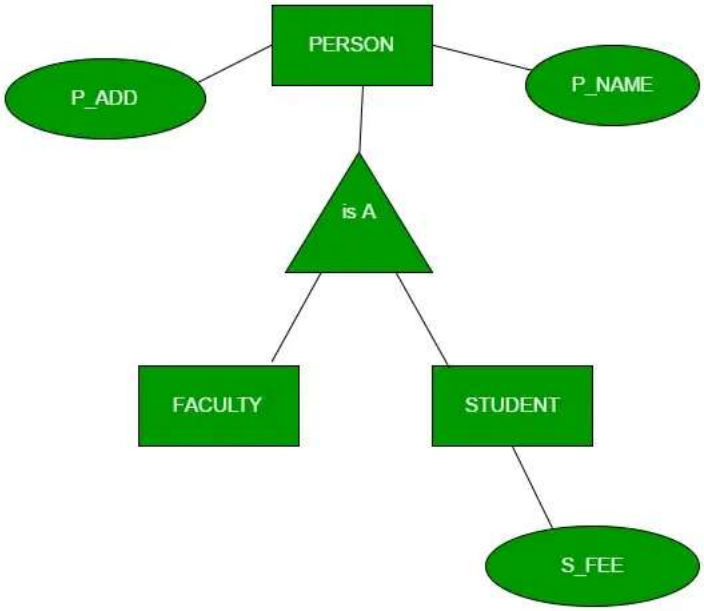
Explanation: In the above tables, the DNO of Table 1 is the foreign key, and DNO in Table 2 is the primary key. DNO = 22 in the foreign key of Table 1 is not allowed because DNO = 22 is not defined in the primary key of table 2. Therefore, Referential integrity constraints are violated here.

Q3.bDemonstrate the concepts of Generalization & Specialization with examples.

Generalization

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher-level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher-level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, and P_ADD become part of a higher entity (PERSON), and specialized attributes like S_FEE become part of a specialized entity (STUDENT).

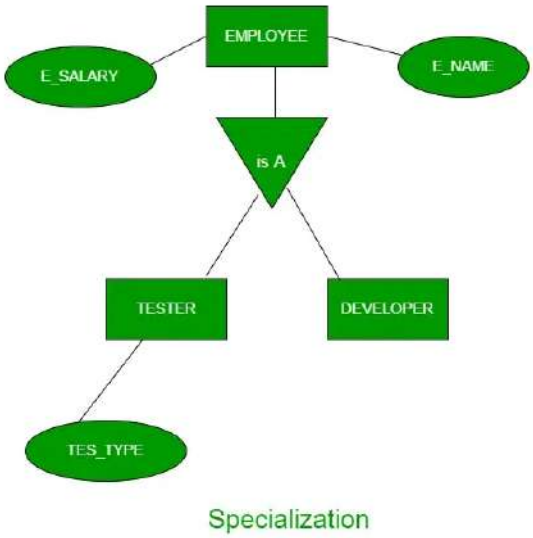
Generalization is also called as ' Bottom-up approach".



Generalization

Specialization

In specialization, an entity is divided into sub-entities based on its characteristics. It is a top-down approach where the higher-level entity is specialized into two or more lower-level entities. For Example, an EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER, etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL, etc. become part of a higher entity (EMPLOYEE), and specialized attributes like TES_TYPE become part of a specialized entity (TESTER).
Specialization is also called as " Top-Down approch".

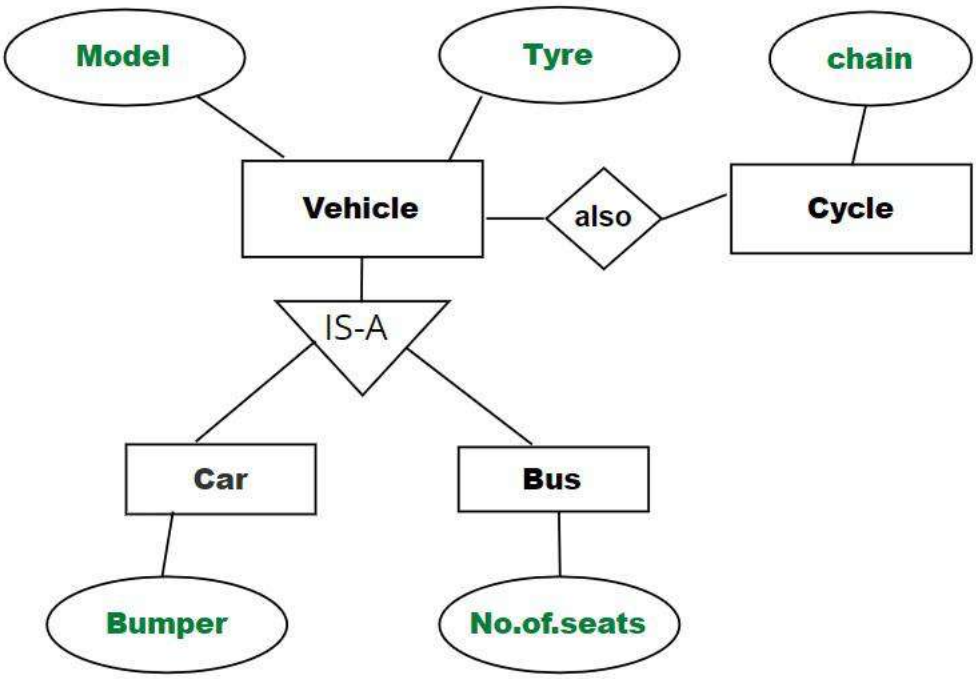


Specialization

Inheritance: It is an important feature of generalization and specialization

- **Attribute inheritance**: allows lower level entities to inherit the attributes of higher level entities and vice versa.
- in diagram: **Car** entity is an inheritance of **Vehicle** entity ,So Car can acquire attributes of **Vehicle** example:car can acquire **Model** attribute of **Vehicle**.

- **Participation inheritance**: In participation inheritance, relationships involving higher level entity set also inherited by lower level entity and vice versa.
- in diagram: Vehicle entity has an relationship with Cycle entity ,So **Cycle entity** can acquire attributes of lower level entities i.e **Car** and **Bus** since it is inheritance of **Vehicle.**



Q3.C c Explain Entity Integrity Constraint & Referential Integrity Constraints? Whyeach of these is important in a database.

Entity integrity constraints:

Entity integrity constraints state that primary key can never contain null value because primary key is used to determine individual rows in a relation uniquely, if primary key contains null value then we cannot identify those rows. A table can contain null value in it except primary key field.

Example:

It is not allowed because it is containing primary key as NULL value.

Student_id	Name	Semester	Age
21CSE101	Ramesh	5th	20
21CSE102	Kamlesh	5th	21
21CSE103	Aakash	5th	22
	Mukesh	5th	20

Referential integrity constraints:

It can be specified between two tables. In case of referential integrity constraints, if a Foreign key in Table 1 refers to Primary key of Table 2 then every value of the Foreign key in Table 1 must be null or available in Table 2.

Example:

Here, in below example Block_No 22 entry is not allowed because it is not present in 2nd table.

Student_id	Name	Semester	Block_No
22CSE101	Ramesh	5th	20
21CSE105	Kamlesh	6th	21
22CSE102	Aakash	5th	20
23CSE106	Mukesh	2nd	22
Block_No	**Block Location**		
20	Chandigarh		
21	Punjab		
25	Delhi		

Importance:

- **Data Consistency**: Ensures that relationships between tables are preserved, and that references are valid. This prevents orphaned records and maintains data accuracy.
- **Preventing Invalid Data**: Helps prevent the entry of data that would violate the established relationships between tables, such as entering an order for a non-existent customer.
- **Enforcing Referential Actions**: Defines what happens when a referenced record is updated or deleted. For instance, it can specify that related orders should be deleted

(cascade delete) or that the foreign key should be set to null (set null) when a customer record is deleted.

Q4.a Consider the Sailors-Boats-Reserves DB described

s (sid, sname, rating, age) b (bid, bname, color)

r (sid, bid, date)

Write each of the following queries in SQL.

1. Find the colors of boats reserved by Alber.

2. Find all sailor ids of sailors who have a rating of at least 8 or reserved boat 103.

3. Find the names of sailors who have not reserved a boat whose name contains the string "storm". Order the names in ascending order.

4. Find the sailor ids of sailors with age over 20 who have not reserved a boat whose name includes the string "thunder".

---

1. Find the colors of boats reserved by Alber.

To find the colors of boats reserved by a sailor named "Alber," you need to join the `Sailors`, `Reserves`, and `Boats` tables. First, identify the `sid` for the sailor named "Alber," and then use it to find the colors of the boats reserved by this sailor.

```
SELECT DISTINCT b.color

FROM Sailors s

JOIN Reserves r ON s.sid = r.sid

JOIN Boats b ON r.bid = b.bid

WHERE s.sname = 'Alber';
```

2. Find all sailor ids of sailors who have a rating of at least 8 or reserved boat 103.

To find all sailor ids of sailors who either have a rating of at least 8 or have reserved boat 103, you need to combine results based on these two conditions.

```
SELECT DISTINCT s.sid
```

```
FROM Sailors s

LEFT JOIN Reserves r ON s.sid = r.sid

WHERE s.rating >= 8 OR r.bid = 103;
```

3. Find the names of sailors who have not reserved a boat whose name contains the string “storm”. Order the names in ascending order.

To find the names of sailors who have not reserved any boat with a name containing "storm," you need to use a subquery to exclude sailors who have reserved such boats.

```
SELECT DISTINCT s.sname

FROM Sailors s

WHERE s.sid NOT IN (

    SELECT r.sid

    FROM Reserves r

    JOIN Boats b ON r.bid = b.bid

    WHERE b.bname LIKE '%storm%'
)

ORDER BY s.sname ASC;
```


4. Find the sailor ids of sailors with age over 20 who have not reserved a boat whose name includes the string “thunder”.

To find the sailor ids of sailors older than 20 who have not reserved a boat with a name containing "thunder," you need a subquery to exclude those who have reserved such boat

```
SELECT DISTINCT s.sid

FROM Sailors s

WHERE s.age > 20 AND s.sid NOT IN (

    SELECT r.sid

    FROM Reserves r
```

```
    JOIN Boats b ON r.bid = b.bid
    WHERE b.bname LIKE '%thunder%'
);
```

Q4.b Discuss the Equijoin & Natural Join with suitable example.

Equijoin and **Natural Join** are both types of joins used in SQL to combine rows from two or more tables based on a related column between them. They serve similar purposes but differ in their specific operations and results.

1. Equijoin

Definition: An Equijoin is a type of join where tables are joined based on the equality of specified columns. It involves a condition where the values in one column from the first table are matched with values in a column from the second table using the equality operator (=).

Example: Consider two tables, Employees and Departments:

Employees Table:

emp_id	emp_name	dept_id
1	Alice	101
2	Bob	102
3	Carol	101

Departments Table:

dept_id	dept_name
101	HR
102	IT
103	Finance

To find the names of employees along with their department names, you can perform an equijoin on the dept_id column:

sql
Copy code

```
SELECT e.emp_name, d.dept_name
FROM Employees e
JOIN Departments d
ON e.dept_id = d.dept_id;
```

Result:

emp_name	dept_name
Alice	HR
Bob	IT
Carol	HR

Explanation: In this equijoin, the dept_id column is used to match rows between the Employees and Departments tables. The equality condition e.dept_id = d.dept_id is applied to combine relevant rows.

2. Natural Join

Definition: A Natural Join automatically joins tables based on columns with the same names and compatible data types in both tables. It eliminates duplicate columns from the result. The join is performed on all columns with the same names in both tables.

Example: Using the same Employees and Departments tables:

Employees Table:

emp_id	emp_name	dept_id
1	Alice	101
2	Bob	102
3	Carol	101

Departments Table:

dept_id	dept_name
101	HR
102	IT
103	Finance

To find the names of employees along with their department names using a natural join:

sql
Copy code

```
SELECT emp_name, dept_name
FROM Employees
NATURAL JOIN Departments;
```

Result:

emp_name	dept_name
Alice	HR
Bob	IT
Carol	HR

Explanation: The NATURAL JOIN automatically joins the Employees and Departments tables on the dept_id column, as it is common to both tables. The result set includes only one dept_id column, with duplicate columns being eliminated.

Key Differences

- **Join Condition**:
- **Equijoin**: Requires an explicit condition for equality using ON clause. You specify which columns to match.

- **Natural Join**: Automatically uses columns with the same names for the join condition. No need to specify the join condition explicitly.
- **Duplicate Columns**:
- **Equijoin**: Both tables' columns included in the join condition appear in the result set unless specified otherwise.
- **Natural Join**: Removes duplicate columns that are used for the join, showing each column only once in the result.
- **Flexibility**:
- **Equijoin**: More flexible because you can choose which columns to join on and how to handle them.
- **Natural Join**: Less flexible as it automatically joins on all columns with the same names and does not provide explicit control over the join conditions.

Q4.c Explain the relational algebra operation for set theory with examples.

Relational algebra operations based on set theory are fundamental in querying and manipulating relational databases. These operations operate on relations (tables) and produce new relations as results. Here's an overview of key relational algebra operations that are derived from set theory, along with examples for each:

1. Union (∪)

Definition: The union operation combines the tuples of two relations, removing duplicates. It requires that both relations have the same schema (i.e., the same number of attributes with the same domain).

Notation:

R∪S

Example:

Consider two relations, **Students1** and **Students2**:

Students1:

student_id	name
1	Alice
2	Bob

Students2:

student_id	name
2	Bob
3	Carol

The union of `Students1` and `Students2` is:

SELECT * FROM Students1

UNION

SELECT * FROM Students2;

Result:

student_id	name
1	Alice
2	Bob
3	Carol

2. Intersection (∩)

Definition: The intersection operation returns the tuples that are common to both relations. Like union, it requires that both relations have the same schema.

Notation: R∩S

Example:

Using the same relations as above:

Students1:

student_id	name
1	Alice
2	Bob

Students2:

student_id	name
2	Bob
3	Carol

The intersection of Students1 and Students2 is:

SELECT * FROM Students1
INTERSECT
SELECT * FROM Students2;

Result:

student_id	name
2	

3. Difference (–)

Definition: The difference operation returns tuples that are present in the first relation but not in the second. Both relations must have the same schema.

Notation:R−S

Example:

Using the same relations as above:

Students1:

student_id	name
1	Alice
2	Bob

Students2:

student_id	name
2	Bob
3	Carol

The difference of **Students1** minus **Students2** is:

```
SELECT * FROM Students1
EXCEPT
SELECT * FROM Students2;
```

Result:

student_id	name
1	Alice

4. Cartesian Product (×)

Definition: The Cartesian product operation combines each tuple of one relation with each tuple of another relation. This operation results in a relation that includes all possible combinations of tuples from the two relations.

Notation:

R×S

Example:

Consider two relations:

Students:

student_id	name
1	Alice
2	Bob

Courses:

course_id	course_name
101	Math
102	Science

The Cartesian product of **Students** and **Courses** is:

```
SELECT * FROM Students
CROSS JOIN Courses;
```

Result:

student_id	name	course_id	course_name
1	Alice	101	Math
1	Alice	102	Science
2	Bob	101	Math
2	Bob	102	Science

5. Rename (ρ)

Definition: The rename operation changes the name of a relation or its attributes. This operation is useful for providing more meaningful names or for resolving naming conflicts in queries involving multiple relations.

Notation: ρnew_name(R)\rho_{new_name}(R)ρnew_name(R) or ρnew_name(A1,A2,…,An)(R)\rho_{new_name(A1, A2, \ldots, An)}(R)ρnew_name(A1,A2,…,An)(R)

Example:

Consider a relation **Employees**:

Employees:

emp_id	emp_name
1	Alice
2	Bob

To rename **Employees** to **Staff** and the attribute **emp_name** to **name**, you can use: SELECT emp_id AS id, emp_name AS name
FROM Employees;

Result:

id	name
1	Alice
2	Bob

Q5.a Explain the Cursor & its properties in embedded SQL with an example.

A cursor is a database object used to retrieve and process rows from a query result set one at a time.

Properties of Cursors

1. **Declare**: Define the cursor with a `DECLARE` statement, specifying the SQL query.
2. **Open**: Use `OPEN` to execute the query and create the result set.
3. **Fetch**: Use `FETCH` to retrieve individual rows from the result set.
4. **Close**: Use `CLOSE` to release resources associated with the cursor.
5. **Deallocate**: Optionally use `DEALLOCATE` to remove the cursor definition and free resources.

Example

Here's a basic example using a cursor in embedded SQL:

```
-- Declare the cursor

DECLARE emp_cursor CURSOR FOR

SELECT emp_id, emp_name, salary FROM Employees;
```



```
-- Open the cursor

OPEN emp_cursor;


-- Fetch rows

FETCH NEXT FROM emp_cursor INTO @emp_id, @emp_name, @salary;


-- Process rows

WHILE @@FETCH_STATUS = 0

BEGIN

    -- Do something with the data (e.g., print it)
```

```
    PRINT 'ID: ' + CAST(@emp_id AS VARCHAR) + ', Name: ' + @emp_name;


    -- Fetch the next row
    FETCH NEXT FROM emp_cursor INTO @emp_id, @emp_name, @salary;
END;


-- Close the cursor
CLOSE emp_cursor;


-- Deallocate the cursor
DEALLOCATE emp_cursor;
```

Q5.b What is a Normalization? Explain the 1NF, 2NF & 3NF with examples.

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. The goal is to ensure that the database structure is efficient and logical, and to eliminate anomalies in data handling. Normalization involves decomposing tables into smaller tables and defining relationships among them based on rules called **normal forms**.

Here's an explanation of the first three normal forms (1NF, 2NF, and 3NF) with examples:

1. First Normal Form (1NF)

Definition: A table is in **First Normal Form (1NF)** if all the columns contain atomic (indivisible) values, and each column contains only one type of data. Additionally, each column must have a unique name, and the order in which data is stored does not matter.

Example:

Consider a table storing information about students and their enrolled courses:

Student_Courses:

student_id	student_name	courses
1	Alice	Math, Science
2	Bob	Science, History

This table is not in 1NF because the **courses** column contains multiple values.

To convert this to 1NF:

Create a separate row for each course.

Student_Courses_1NF:

student_id	student_name	course
1	Alice	Math
1	Alice	Science
2	Bob	Science
2	Bob	History

2. Second Normal Form (2NF)

Definition: A table is in **Second Normal Form (2NF)** if it is in 1NF and all non-key attributes are fully functionally dependent on the entire primary key. In other words, there should be no partial dependency of any column on a subset of a composite primary key.

Example:

Consider a table with the following schema and data:

Orders:

order_id	product_id	product_name	quantity
1	101	Laptop	2
1	102	Mouse	1
2	101	Laptop	1

Here, **order_id** and **product_id** together form the composite primary key. **product_name** is dependent only on **product_id** and not on **order_id**.

To convert this to 2NF:

Separate the product information into its own table.

Orders:

order_id	product_id	quantity
1	101	2
1	102	1
2	101	1

Products:

product_id	product_name
101	Laptop
102	Mouse

3. Third Normal Form (3NF)

Definition: A table is in **Third Normal Form (3NF)** if it is in 2NF and all the attributes are functionally dependent only on the primary key, and not on any other non-key attributes. This means there should be no transitive dependency.

Example:
Consider a table with the following schema:
Employees:

emp_id	emp_name	department	department_manager
1	Alice	HR	John Doe
2	Bob	IT	Jane Smith

In this table, **department_manager** depends on **department**, which is a non-key attribute.
To convert this to 3NF:
Remove the transitive dependency by creating a separate table for department details.
Employees:

emp_id	emp_name	department
1	Alice	HR
2	Bob	IT

Departments:

department	department_manager
HR	John Doe
IT	Jane Smith

Q5.C Explain informal design guidelines for relational schema design.

1. Minimize Redundancy

Guideline: Avoid duplicating data across tables. Redundant data increases storage requirements and can lead to inconsistencies. Ensure that each piece of information is stored only once.

- **Example**: Instead of storing a customer's address in every order record, store the address in a separate `Customers` table and reference it in the `Orders` table.

2. Avoid Insertion, Update, and Deletion Anomalies

Guideline: Design the schema to prevent anomalies that can occur when inserting, updating, or deleting records. This typically involves ensuring that the schema is normalized to a suitable level (usually 3NF).

- **Example**: If you have a table where updating a customer's address could require multiple updates due to redundant storage, it's better to have a separate `Addresses` table and link it with foreign keys.

3. Ensure Data Integrity

Guideline: Use constraints to ensure that the data in the database is accurate and reliable. This includes primary keys to uniquely identify records, foreign keys to maintain referential integrity, and other constraints to enforce valid data.

- **Example**: Use foreign keys to ensure that every `order` in the `Orders` table refers to a valid `customer` in the `Customers` table.

4. Design for Flexibility and Simplicity

Guideline: The schema should be designed to be simple and flexible. Avoid overly complex designs and ensure the schema can accommodate future changes with minimal modifications.

- **Example**: Instead of combining multiple types of information into one table, use separate tables with clear relationships. For example, use separate tables for `Employees` and `Departments` with a clear foreign key relationship.