DBMS (unit 1+3) SQL.

||| ① Explain the three schema architecture with the help of a neat diagram. (1st)

| ② Explain the following with examples.
a) ALTER TABLE command
b) GROUP BY – HAVING clause.

| ③ Explain the aggregate operators in SQL with example.

||| ④ Bring out the important advantages and disadvantages of DBMS over file system. (1st)

| ⑤ Explain with example any four column constraints used in SQL.

|| ⑥ concept of primary and foreign keys with examples. (wrt to relational data model)

||| ⑦ Explain various JOIN operations in SQL with example.
(wt do u mean by JOIN in sql)

||| → complex SQL queries with examples. (INSERT UPDATE DELETE)

| ⑧ With a neat block diagram explain the DBMS architecture. Also discuss 3 different levels of abstraction. (1st)

|| ⑨ Explain with examples concepts of assertions & triggers.

⑩ Describe the six clauses of SQL query statements and show that expression of relations can be specified.

| ⑪ Data types allowed for SQL attributes. examples

| ⑫ Briefly explain views along with syntax. Discuss the problems that may rise when one attempts to update a view.

| ⑬ define terms with example each DBMS|metadata (1st) schema
Domain | Relation schema | Relation state | Relation database

⑭ Nested query with example.

⑮ DROP | DELETE | SELECT commands in sql.

|| ⑯ Sailors (Sid, Sname, Rating, age)
Boats (Bid, Bname, color)
Reserves (Sid, Bid, Date)

① find names of (boat) sailors who have reserved red boat

② Find colors of boat reserved by Andy.

11 (17) Project (P_NO, P_Name, P_Incharge)

Employee (E_NO, E_name)

Assigned_to (P_NO, E_NO)

① List details of employees who are working on all the projects

② List E_NO of employees who are not taking work on the 2K project

③ List the names of employees who are working in the same project as employee named 'Tom'

④ List the names of employees who are not working in any project.

(18) Characteristics of a database management system. (1st)

(19) Explain different phase of data base design. (1st)

DBMS(DATABASE MANAGEMENT SYSTEM)

IMPORTANT QUESTIONS AND ANSWERS

KRUTHIK B

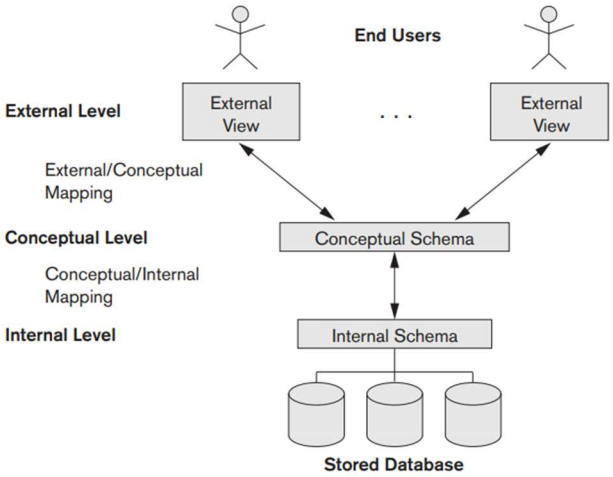
1DA19CS077

B SECTION

UNIT 1

1)

The goal of the three-schema architecture, illustrated in Figure in the below given figure, is to separate the user applications from the physical database.



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.

The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints . This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

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 that user group.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. In the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval then the data must be extracted from the stored database and be reformatted to the users view (This process of transforming requests and results between levels are called **mappings**).

KRUTHIK B (1DA19CS077 B SECTION)

4)

ADVANTAGES

Data redundancy(repetition) and inconsistency

The file system cannot control redundancy of data as each user defines and maintains the needed files for a specific application to run. There may be a possibility that two users are maintaining same files data for different applications. Hence changes made by one user does not reflect in files used by second users, which leads to inconsistency of data. Whereas DBMS controls **redundancy by maintaining a single repository** of data that is defined once and is accessed by many users. As there is no or less redundancy, data remains consistent.

Data sharing

File system does not allow sharing of data or sharing is too complex. Whereas in DBMS, data can be shared easily due to **centralized system.**

Data concurrency

Anomalies occur when changes made by one user gets lost because of changes made by other user. File system does not provide any procedure to stop anomalies. Whereas DBMS provides a **locking system to stop anomalies** to occur.

Data searching

For every search operation performed on file system, a different application program has to be written. While DBMS provides inbuilt searching operations. User only have to write a small **query** to retrieve data from database.

Data integrity

There may be cases when some constraints need to be applied on the data before inserting it in database. The file system does not provide any procedure to check these constraints automatically. Whereas DBMS maintains data integrity **by enforcing user defined constraints** on data by itself.

System crashing

System crash is a bane in case of file systems because once the system crashes, there will be no recovery of the data that's been lost. A DBMS will have the **recovery manager** which retrieves the data making it another advantage over file systems.

DISADVANTAGES

Cost of Hardware and Software
To store a huge amount of data, one needs a huge amount of space. Additionally, it will require more memory and fast processing power to run the DBMS.

Complexity
A DBMS fulfills lots of requirements and it solves many problems related to databases.

Database Failure
Data is the key for any organization, if data is lost then the whole organization will collapse.

Huge Size
DBMS is made to handle extremely huge data and queries, but due to its complexity, DBMS has become huge in size. Also, it becomes bigger in size as data is fed into it. As a result, it requires lots of space and memory to run its application efficiently.

KRUTHIK B (1DA19CS077 B SECTION)

13)

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

The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary it is called **meta-data**.

In a data model, it is important to distinguish between the description of the database and the database itself. The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently. A **displayed schema is called a schema diagram**

A **relation schema** R, denoted by R(A1, A2, … , An), is made up of a relation name R and a list of attributes, A1, A2, … , An. Each attribute Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai). A relation schema is used to describe a relation; R is called the name of this relation.

A **domain** D is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.

KRUTHIK B (1DA19CS077 B SECTION)

18)

Real World Entity

DBMS these days is very realistic and real-world entities are used to design its architecture. Also, behavior and attributes are used by DBMS.

Self-Describing Nature

A DBMS should be of Self- Describing nature as it not only contains the database itself but also the metadata. A metadata (data about data) defines and describes not only the extent, type, structure, and format of all data but also relationship between data.

Insulation Between Data and Program

Program-data independence provides a big relief to database users. In traditional file management system, structure of data files was defined in the

application programs so the user had to change all the programs that are using that particular data file.
But in DBMS, structure of data files is not stored in the program but it is stored in system catalogue.

Multiple Views
Users can have multiple views of database depending on their department and interest. DBMS support multiple views of database to the users. For example, a user of the teaching department will have different view and user of hostel department will have different. This feature helps users to have somewhat security because users of other departments cannot access their files.

Query Language
Queries are used to retrieve and manipulate data but DBMS is armed by a strong query language that makes it more effective and efficient. Users have the power to retrieve any kind of data they want from the database by applying different sets of queries.

KRUTHIK B (1DA19CS077 B SECTION)

19) No
9) No
8) No

KRUTHIK B (1DA19CS077 B SECTION)

UNIT 3

2)

<u>ALTER TABLE</u>

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ADD COLUMN ALTER TABLE *table_name*
ADD *column_name datatype*;

This sql adds ‘Email column to the Customer Table’

ALTER TABLE Customers
ADD Email varchar(255);

DROP COLUMN ALTER TABLE *table_name*
DROP COLUMN *column_name*;

This sql deletes 'Email column to the Customer Table'

ALTER TABLE Customers
DROP COLUMN Email;

MODIFY COLUMN ALTER TABLE *table_name*
MODIFY *column_name datatype*;

GROUP BY _ HAVING clause

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

SYNTAX SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition;*

The SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

SELECT COUNT(CustomerID),Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;

3)

1)The **MIN()** function returns the smallest value of the selected column.

MIN() Syntax

SELECT MIN(*column_name*)
FROM *table_name*;

The following SQL statement finds the price of the cheapest product:

SELECT MIN(Price) AS SmallestPrice
FROM Products;

2)The **MAX()** function returns the largest value of the selected column.

MAX() Syntax

SELECT MAX(*column_name*)
FROM *table_name*;

The following SQL statement finds the price of the most expensive product:

SELECT MAX(Price) AS LargestPrice
FROM Products;

3)The **COUNT()** function returns the number of rows that matches a specified criterion.

COUNT() Syntax

SELECT COUNT(*column_name*)
FROM *table_name*;

The following SQL statement finds the number of products:

SELECT COUNT(ProductID)
FROM Products;

4)The **AVG()** function returns the average value of a numeric column.

AVG() Syntax

SELECT AVG(*column_name*)
FROM *table_name*;

The following SQL statement finds the average price of all products:

SELECT AVG(Price)
FROM Products;

5)The **SUM()** function returns the total sum of a numeric column.

SUM() Syntax

SELECT SUM(*column_name*)
FROM *table_name*;

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

SELECT SUM(Quantity)
FROM OrderDetails;

KRUTHIK B (1DA19CS077 B SECTION)

5) QUESTION

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

SYNTAX

CREATE TABLE *table_name* (
  column1 datatype constraint,
  column2 datatype constraint,
  column3 datatype constraint,
  ....
);

NOT NULL - Ensures that a column cannot have a NULL value

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

CREATE TABLE Persons (
  ID int NOT NULL,

```
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int
);
```

UNIQUE - Ensures that all values in a column are different

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL UNIQUE,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int
);
```

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   PRIMARY KEY (ID)
);
```

FOREIGN KEY - Prevents actions that would destroy links between tables

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
```

```
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

DEFAULT - Sets a default value for a column if no value is specified

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'Sandnes'
);
```

KRUTHIK B (1DA19CS077 B SECTION)

6)

*The **PRIMARY KEY** constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key and two or more columns together form a composite primary key of that table

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   PRIMARY KEY (ID)
);
```

Object Type **TABLE** Object **PERSONS**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
PERSONS	ID	Number	-	-	0	1	-	-	-
	LASTNAME	Varchar2	255	-	-	-	-	-	-
	FIRSTNAME	Varchar2	255	-	-	-	✓	-	-
	AGE	Number	-	-	0	-	✓	-	-
									1 - 4

*A **FOREIGN KEY** is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created by referencing ID from Persons table:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(ID)
);
```

Object Type **TABLE** Object **ORDERS**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
ORDERS	ORDERID	Number	-	-	0	1	-	-	-
	ORDERNUMBER	Number	-	-	0	-	-	-	-
	PERSONID	Number	-	-	0	-	✓	-	-
									1 - 3

KRUTHIK B (1DA19CS077 B SECTION)

7)

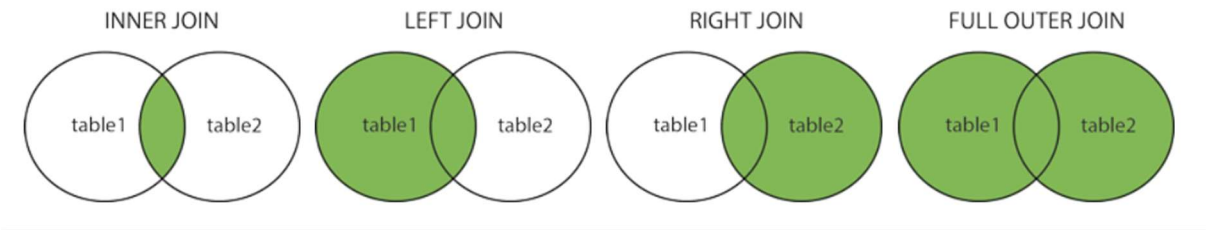
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

INNER JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

The following SQL statement joins Customers and orders on the basis of same CustomerID

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

LEFT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

The following SQL statement will select all customers, and any orders they might have:

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;

RIGHT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

The following SQL statement will return all employees, and any orders they might have placed:

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;

FULL OUTER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name* = *table2.column_name*
WHERE *condition*;

The following SQL statement selects all customers, and all orders:

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

KRUTHIK B (1DA19CS077 B SECTION)

10)

There are generally five kinds of SQL Clauses in MySQL Server. They are listed as follows:

1)**WHERE Clause**

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

SELECT * FROM Customers
WHERE Country='Mexico';

2)**ORDER BY clause**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
ORDER BY *column1, column2, ...* ASC|DESC;

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

SELECT * FROM Customers
ORDER BY Country;

3)**HAVING Clause**

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition;*

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

4)TOP Clause

5)**<u>GROUP BY Clause</u>**

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
GROUP BY column_name(s)
ORDER BY column_name(s);
```

The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

KRUTHIK B (1DA19CS077 B SECTION)

11)

The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.

NUMBER

1) ***NUMBER (p,s):*** Numbers with magnitude between 1.0×10^{-130} & 9.99×10^{125} , p is the precision ranging from 1 to 38 & s is scale, *the number* of digits after the decimal point ranging from -84 to 127.

2) ***FLOAT(b):*** Specifies a floating point number with binary precision b ranging fron 1 to 122.

3) ***INTEGER or INT, and SMALL INT:*** Used to specify an integer number.

4) ***DECIMAL (p,s):*** Used to specify a decimal number.p total number of digits before point and s represent decimal numbers after point.

CHARACTER-STRING

1)***Fixed length***—CHAR(*n) or* CHARACTER(*n), where n is the number of characters or size*

*2)**Varying length***— VARCHAR(*n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is* the maximum number of characters.

BIT-STRING

1)***Fixed length n****—BIT(n)*

*2)**Varying length***—BIT VARYING(*n),*

where n is the maximum number of bits. The default for n, the length of a character string or bit string, is 1.

BOOLEAN data type has the traditional values of **TRUE** or **FALSE.** In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

DATE data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD. But in the latest version of oracle 10g express edition the date has 9 positions DD-MON-YY.

TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS

KRUTHIK B (1DA19CS077 B SECTION)

14)

Considering the following database

[Sailors(sid: integer, sname: string, rating: integer, age: real)

Boats(bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: date)]

NESTED QUERY

Is a query that has another query embedded within it & the embedded query is called a **Sub query .** That other query is called the **outer query.** A sub query usually appears in the "WHERE" clause of a query but it cal also be written in the "FROM" clause

1) Find the names of sailors who have reserved boat 103.

```
SELECT S.SNAME
FROM SAILORS S
WHERE S.SID IN (  SELECT R.SID
                  FROM RESERVES R
                  WHERE R.BID=103);
```

2) Find the names of sailors who have reserved a red boat.

```
SELECT S.SNAME
FROM SAILORS S
WHERE S.SID IN (  SELECT R.SID
                  FROM RESERVES R
                  WHERE R.BID IN  ( SELECT B.BID
                                    FROM BOATS B
                                    WHERE B.COLOR = "RED"));
```

(Explain the above queries in ur own words)

KRUTHIK B (1DA19CS077 B SECTION)

->and 15)

The **INSERT INTO** statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

The **UPDATE** statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;

The **DELETE** statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table_name*;

The **SELECT** statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT * FROM *table_name*;

KRUTHIK B (1DA19CS077 B SECTION)

16)

Sailors(sid: integer, sname: string, rating: integer, age: real)

Boats(bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: date)

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

```
SELECT Sailors.SNAME
FROM SAILORS
WHERE Sailors.SID IN (  SELECT Reserves.SID
                        FROM RESERVES
                        WHERE Reserves.BID IN  ( SELECT Boats.BID
                                                 FROM BOATS
                                                 WHERE Baots.COLOR = "RED"));
```

Find the colors of boats reserved by Andy.

SELECT Boats.color

FROM SAILORS , BOATS , RESERVES

WHERE Sailors.SID = Reserves.SID AND Boats.BID = Rreserves.BID AND Sailors.SNAME='ANDY';

17) Write on ur own

12) No

KRUTHIK B (1DA19CS077 B SECTION)

DBMS (unit 1 half / unit 2 / unit 5)

Library info system (4)
Bank database

1. Draw the ER diagram for student database / hospital database considering at least minimum 5 entities.
2. X What is data independence? List and explain different types of data independences.
3. 1 Define entity, entity set, attribute with respect to ER model. List different types of attributes along with their symbols.
4. unit 2 II Define with an example each (Domain / Relation schema / Relation state / Relation database schema)
5. Define (weak entity / strong entity / cardinality ratio / participation constraints / composite attributes)
6. Explain with sketch different phases of database design.

7. 1 Briefly discuss the different types of UPDATE operations on relational database. Show with an ex constraint violated in each of the update operation.
8. II Explain any 2 set operations in relational algebra. Illustrate with some example. [U, ∩, −]
9. Discuss entity and referential integrity constraints. Why are they imp
10. Explain domain const, key const and constraints on NULL values.
11. II wrt database relation model explain PK & FB with suitable ex.
12. III DISCUSS the opn Illustrate each of them with example.
    SELECT / PROJECT / DIFFERENCE / UNION / INTERSECT
13. wt do you mean by JOIN? Explain various JOIN opn in SQL with an ex. relational model. (θ join required)
14. Different types of constraints on relationship type? with ex

15. Explain in brief transaction schedules based on serializability and Recoverability.
16. 1 Explain how to resolve deadlock and starvation problems. approaches to deal with it.
17. 1 Discuss the Acid properties.
18. What is schedule (history)? different concepts of strict schedulers.
19. II problem

c)	Consider the following schema Sailors(Sid,Sname,Rating,age) Boats(Bid,Bname,color) Reserves(Sid,Bid,day) Write queries in relational algebra 1. Write the names of sailor who have reserved boat number 103. 2. Display names of sailor who have reserved green and red boat. 3. Find the names of sailors who have not reserved a red boat. 4. Find all sailors with a rating above 7. 5. Find the colors of boats reserved by Andy.	5 M

a) Consider the following schema for a company database.

Employee (Name, SSN, Address, Sex, Salary, Dno)
Department (Dname, Dnumber, MGR SSN, MGRSTART DATE)
Dept_Location (Dnumber, Dlocations)
Project (Pname, Pnumber, Plocation, Dnum)
Works_on (ESSN, Pno, Hours)
Dependent (ESSN, Dependent_name, Sex, Ddate, Relationship)

Write the queries in relational algebra to

1. Retrieve the name and address of all employees who work for the research department
2. Find the name of employees who work on all projects controlled by department number 5.
3. List all projects on which employee Smith is working.
4. Retrieve the names of employees who have no dependents.

b) Consider the following schema

Sailors(Sid,Sname,Rating,age)

Boats(Bid,Bname,color)

Reserves(Sid,Bid,day)

Write queries in relational algebra

i. Write the names of sailor who have reserved boat number 103.
ii. Display names of sailor who have reserved green and red boat.
iii. Display the name of sailor and boat reserved on Thursday.

b) Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

1. r1(x); r3(x); w1(x); r2(x); w3(x);
2. r1(x); r3(x); w3(x); w1(x); r2(x);
3. r3(x); r2(x); w3(x); r1(x); w1(x);
4. r3(x); r2(x); r1(x); w3(x); w1(x);

DATABASE MANAGEMENT SYSTEM

(Important Questions and Answers)

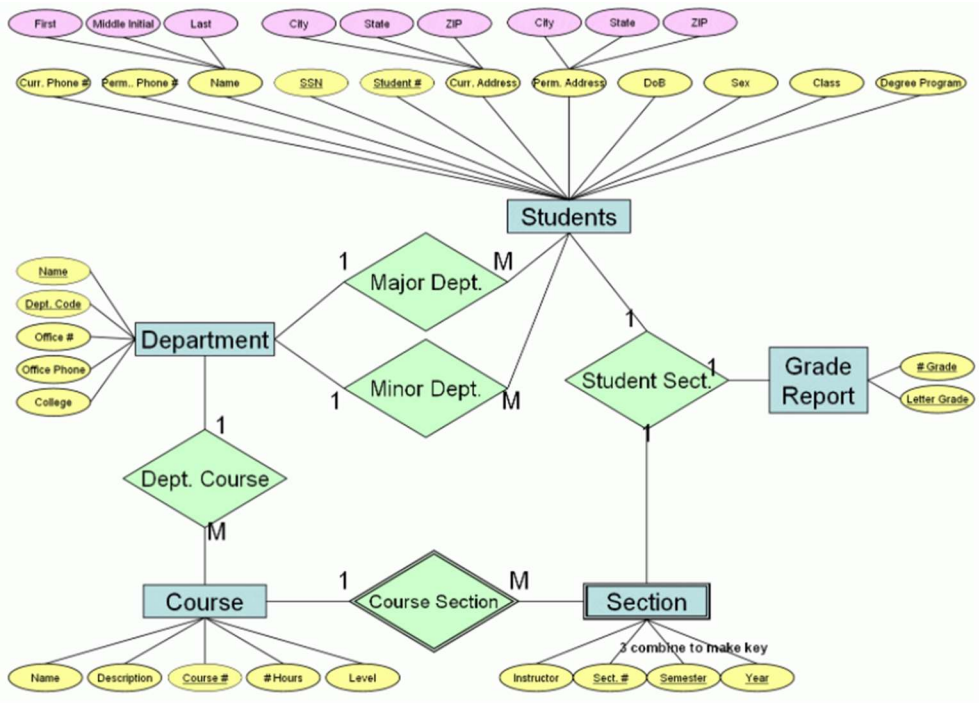
KRUTHIK B

1DA19CS077

B SECTION B2 BATCH

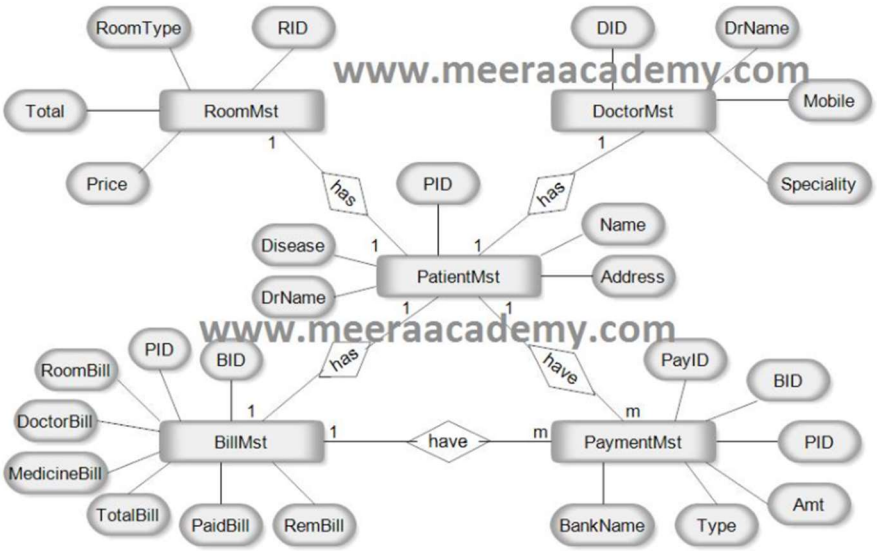
1)

Student Database



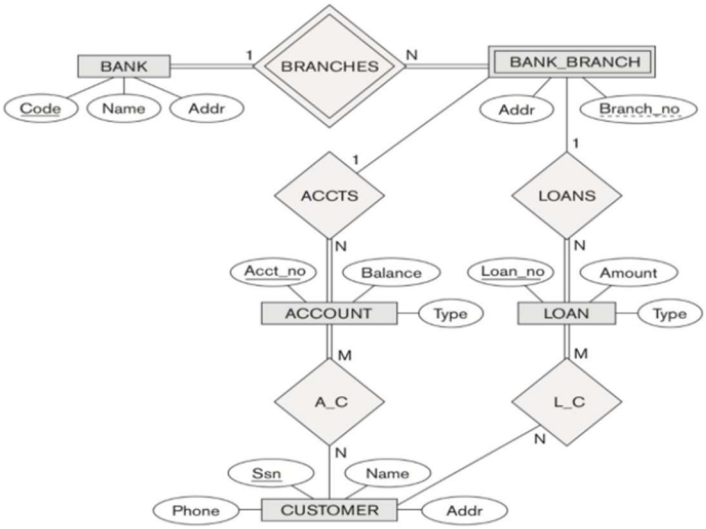
(U can write how many attributes u want)

Hospital Database



www.meeraacademy.com

Bank Database



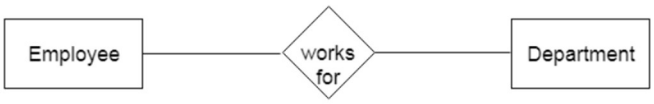
Library Database

KRUTHIK B(1DA19CS077 B SECTION)

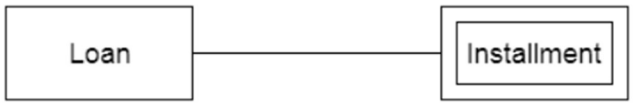
2) No there in syllabus

3) and 5)

Entity(Strong Entity) An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles. Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



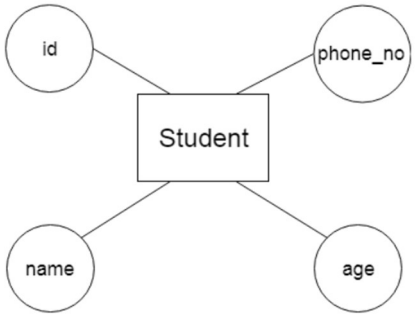
Weak Entity An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



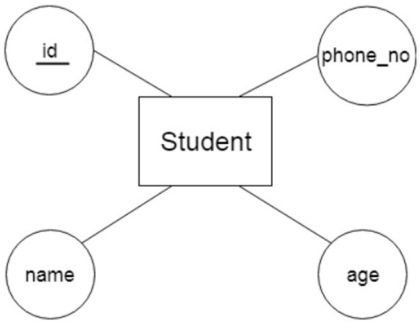
Entity Set Attributes are associated with an entity set. Attributes describe the properties of entities in the entity set.Based on the values of certain attributes,

an entity can be identified uniquely. An entity set if of two types Strong and Weak entity.

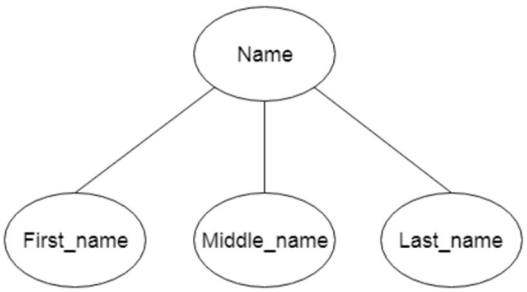
Attribute The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute. **For example,** id, age, contact number, name, etc. can be attributes of a student.



Key Attribute The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



Composite Attribute An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



Multivalued Attribute An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to

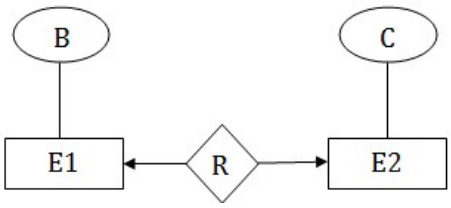
represent multivalued attribute. **For example,** a student can have more than one phone number.

Derived Attribute An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse. **For example,** A person's age changes over time and can be derived from another attribute like Date of birth.

Cardinality Ratio

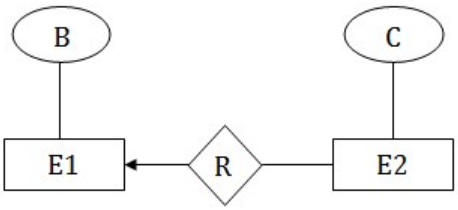
One-to-one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



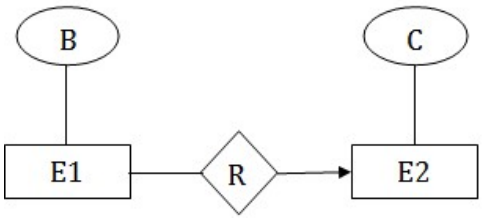
One-to-many

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



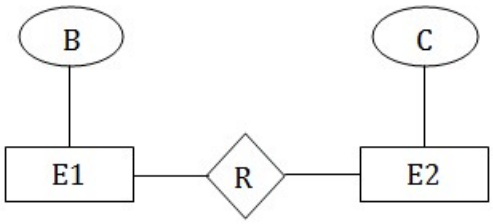
Many-to-one

In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



Many-to-many

In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.



Participation constraints

Participation constraints define the least number of relationship instances in which an entity must compulsorily participate. Is of two types ,

1)Total participation

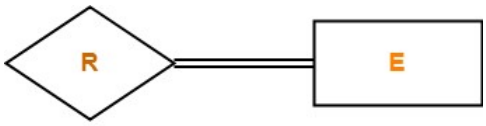
2)Partial participation

Total Participation

->It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set.

->That is why, it is also called as **mandatory participation.**

->Total participation is represented using a double line between the entity set and relationship set.



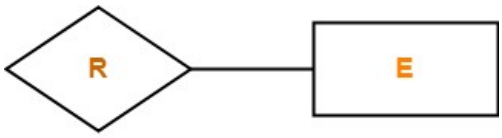
Total Participation

Partial Participation

->It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set.

->That is why, it is also called as **optional participation.**

->Partial participation is represented using a single line between the entity set and relationship set.



Partial Participation

KRUTHIK B(1DA19CS077 B SECTION)

5)

Domain: A (usually named) set/universe of *atomic* values, where by "atomic" we mean simply that, from the point of view of the database, each value in the domain is indivisible (i.e., cannot be broken down into component parts). Examples of domains

SSN: string of digits of length nine

Name: string of characters beginning with an upper case letter

Relational Schema: used for describing (the structure of) a relation. E.g., R(A_1, A_2, ..., A_n) says that R is a relation with *attributes* A_1, ... A_n. The **degree** of a relation is the number of attributes it has, here *n*.

Example: STUDENT (Name, SSN, Address)

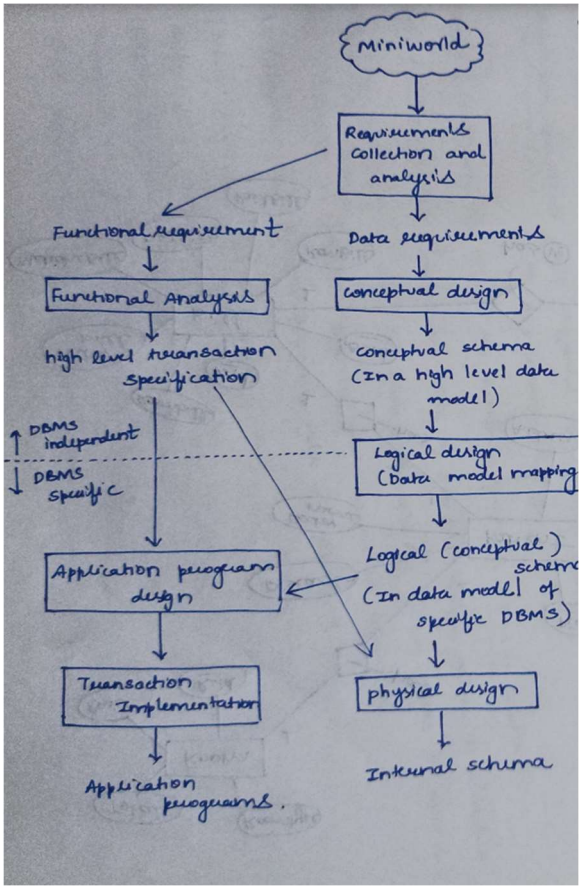
Relational Database: A collection of **relations**, each one consistent with its specified relational schema. A collection of the whole database.

Relational database state: is a union of all the individual relation states, whenever the database is changed a new state arise. Relational state is also cartesian product of degree n on domain.

KRUTHIK B(1DA19CS077 B SECTION)

6)

Phases of Database Design
Database design is made up of three main phases, namely conceptual,logical, and physical design.
Conceptual database design to build the conceptual representation of the database, which includes identification of the important entities,relationships, and attributes.
Logical database design to translate the conceptual representation tothe logical structure of the database, which includes designing the relations.
Physical database design to decide how the logical structure is to bephysically implemented (as base relations) in the target DatabaseManagement System (DBMS).



Explain the flow diagram in ur own words

KRUTHIK B(1DA19CS077 B SECTION)

7)

Four basic update operations performed on relational database model are Insert, update and delete.

The Insert Operation: The **Insert operation** provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Insert can violate any of the four types of constraints.

1)Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.

2)Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).

3)Entity integrity can be violated if any part of the primary key of the new tuple t is NULL.

4)Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

(Ways to deal with it

If an insertion violates one or more constraints, the default option is to reject the insertion.

Give user opportunity to try again with different attribute values.)

The Delete Operation: The **Delete operation** can violate only referential integrity constraint. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database.

(Ways to deal with it: Several options are available if a deletion operation causes a violation.

The first option, called **restrict,** is to reject the deletion.

The second option, called **cascade,** is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted.

The third option, called **set null or set default,** is to modify the referencing attribute values that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple.

Combinations of these three options are also possible.)

The Update Operation: The **Update (or Modify) operation** is used to change the values of one or more attributes in a tuple (or tuples) of some relation *R. It is necessary to specify a condition* on the attributes of the relation to select the tuple (or tuples) to be modified.

Key constraint violation: The primary key is changed so as to become same as another tuple‘s key.

Referential integrity violation: Foreign key is changed and new one refers to nonexistent tuple.

Primary key is changed and now other tuples that had referred to this one violate the constraint.

(**Ways to deal with it**: Modifying a primary key value is similar to deleting one tuple and inserting another in its place because we use the primary key to identify tuples. Hence, the solutions discussed with INSERT operation & DELETE operation are used to deal with Update Operation.)

KRUTHIK B(1DA19CS077 B SECTION)

9) and 10)

Mainly Constraints on the relational database are of 4 types:

1. Domain constraints
2. Key constraints
3. Entity Integrity constraints
4. Referential integrity constraints

1. Domain constraints :

1. Every domain must contain atomic values(smallest indivisible units) it means composite and multi-valued attributes are not allowed.
2. We perform 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 cant give it values other then 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 :

1. These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
2. A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as 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.
3. Null values are not allowed in the primary key, hence Not Null constraint is also a part of 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 first and the last tuple has the same value in EID ie 01, so it is violating the key constraint.

3. Entity Integrity Constraints :

1. Entity Integrity constraints says that no primary key can take NULL value, since using 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 primary key, and the primary key cant take NULL values but in the third tuple, the primary key is null, so it is a violating Entity Integrity constraints.

4. Referential Integrity Constraints :

1. The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
2. This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have 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.
3. 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, DNO of the first relation is the foreign key, and DNO in the second relation is the primary key. DNO = 22 in the foreign key of the first table is not allowed since DNO = 22
is not defined in the primary key of the second relation. Therefore, Referential integrity constraints is violated here

(In 10th question explain that Null values cannot be taken as Primary Key shd not be null and it also effects the foreign key)

(In 9th question explain that PK and FK must be must be off same values in R1 and R2)

KRUTHIK B(1DA19CS077 B SECTION)

11) and 8)

The SELECT Operation (Represented by Sigma)

SELECT a subset of tuples from R that satisfy a selection condition.

σ<selection condition>(R1)

σ(DNO=4 and SALARY >25000) or (DNO=5 and SALARY >30000) (EMPLOY EE)

Here if the first condition or the second condition shd be true in order to select an EMPLOYEE with satisfying conditions.

The PROJECT Operation (Represented by pi)

PROJECT some columns (attributes) from the table (relation) and discard the other columns.

R2 ← π<attribute list>(R1)

R2 ← π(LNAME, SALARY)(EMPLOY EE)

R2 has only the attributes in < attribute list > with the same order as they appear in the list.

degree(R2) = |< attribute list >|

The RENAME operation (Represented by rho)

ρTEMP (σDNO=5(EMPLOY EE))

ρR(F N, LN, SALARY)(πF NAME, LNAME, SALARY (TEMP))

Here , the employee name whose DNO is five is selected and stored in a relation named TEMP and in next line his first name, last name and salary are stored in a relation R with attributes FN,LN,SALARY.

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.*

For ∪, ∩, -, the operand relations R(A1, A2, ..., An) and S(B1, B2, ..., Bn) must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, dom(A_i) = dom(B_i) for i=1, 2, ..., n. This condition is called union compatibility. The resulting relation for ∪, ∩ or - **<u>has the same attribute names as the first operand</u>** relation R (by convention).

(Write examples from below table)

Figure 7.11 Illustrating the set operations union, intersection, and difference. (a) Two union compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∪ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

CARTESIAN PRODUCT: It is denoted by ×. This is also a binary set operation, but the relations on which it is applied do not have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set). In general, the result of R(A1, A2, … , An) × S(B1, B2, … , Bm) is a *relation Q with degree n + m attributes Q(A1, A2, … , An, B1, B2, … , Bm), in that order.*

Example: Combine each DEPARTMENT tuple with the EMPLOYEE tuple of the manager.

DEP_EMP ← DEPARTMENT X EMPLOYEE

DEPT_MANAGER ← $\sigma_{MGRSSN=SSN}$(DEP_EMP)

KRUTHIK B(1DA19CS077 B SECTION)

13)

The JOIN Operation: It is denoted by ⋈, is used to combine *related tuples from two relations* into single "longer" tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

There are two types of joins

INNER JOIN: THETA JOIN, EQUI JOIN & NATURAL JOIN

OUTER JOIN: LEFT OUTER JOIN, RIGHT OUTER JOIN & FULL OUTER JOIN

Consider the following relational database

OUTER JOINS

Consider the relational database:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Left Outer Join (A ⟕ B): This returns all the rows from the table on the left even if no matching rows have been found in the table on the right. When no matching record found in the table on the right, NULL is returned.

EMPLOYEE ⟕ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

Right Outer Join (A ⟖ B): This returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Full Outer Join (A ⟗ B): In a **FULL OUTER JOIN** , all tuples from both relations are included in the result, irrespective of the matching condition.

EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

INNER JOINS

NATURAL JOIN: Denoted by *. NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project. In the following example, first we rename the Dnumber attribute of DEPARTMENT to Dnum—so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN

PROJ_DEPT ← PROJECT * $\rho_{(Dname, Dnum, Mgr_ssn, Mgr_start_date)}$ (DEPARTMENT)

EQUI JOIN: The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUI JOIN.

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

THETA JOIN: A general join condition is of the form
<condition> AND <condition> AND … AND <condition>

Where each <condition> is of the form $A_i \vartheta B_j$, *A_i is an attribute of R, B_j is an attribute* of *S, A_i and B_j have the same domain, and ϑ (theta) is one of the comparison* operators {=, <, ≤, >, ≥, ≠}.

A JOIN operation with such a general join condition is called a THETA JOIN.

KRUTHIK B(1DA19CS077 B SECTION)

11)

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key: The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

- The value of the Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO is the candidate key for relation STUDENT.
- The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.
- No of candidate keys in a Relation are nC(floor(n/2)),for example if a Relation have 5 attributes i.e. R(A,B,C,D,E) then total no of candidate keys are 5C(floor(5/2))=10.

Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

Foreign Key: If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it. For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For Example, STUD_NO in STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuples. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

KRUTHIK B(1DA19CS077 B SECTION)

DBMS unit (5 and 4)

Unit 4

(1)

1 (1) what is need for normalization? Explain 1Nf 2Nf 3Nf with ex

1 (2) consider the universal relation R = {A to J} and the set of functional dependencies. F = {(A,B → C), (A → (D,E)), (B → F), (F → (G,H)), (D → (I,J))} what is the Key of R

Decompose the R to 2Nf and 3Nf

1 (3) Car_Sale (car_no, date_sold, salesman_no, comission, Discount)

[assume a car can be sold by multiple sales man and hence pK is {car_no, salesman_no})

Additional dependencies are,

(discount & salesman_no) → comission

i) Is the relation 1nf, 2nf, 3nf? why or why not.

ii) how would we normalize this completely.

1 (4) Define Boyce-codd normal form. How does it differ from 3nf? why is it considered a stronger form of 3nf.

11 (5) state the informal guidelines for relation schema design. Illustrate with example. (need for normalizn)

1 (6) Explain multi valued dependencies, 4nf, join dependencies and fifth normal form with examples.

(7) Explain informal design guidlines used as measures to determine the quality of relation schema design.

(8) Boyce codd form with example. (4nf with ex)

(9) Let R = {SSN, Ename, Pno, Pname, Ploc, Hours} and D = {R1, R2, R3}

R1 => EMP = {ssn, Ename}

R2 => Project = {Pname, Pno, Ploc}

R3 => Works_On = {ssn, Pno, Hours}

The following functional dependency holds on relation R

F = {ssn → Ename, Pno → {Pname, Ploc}, {ssn, Pno → Hours}}

→ Prove the decomposition of relation (R) has the lossless join property.

unit 5

11 (1) Explain a mechanism for concurrency control that uses time stamping with the help of of an ex.

1 (2) Discuss the problems of deadlock and starvation and different approaches to deal with these problems.

(3) Explain multi granularity locking with example and also explain under wt circumstances is it used.

11 (4) Prove that strict (2) phase locking protocol guarantees [strict schedule] with (ex.) and advantages.

11 (5) ACID properties.

1 (6) Schedule (history) ? define concepts of strict schedules.

11 (7) time stamp ordering algo with example.

11 (8) Explain serializability with ex. How can it be ensured ? Do we need to restrict concurrent execution to ensure serializibility ? Justify.

(9) Lock based concurrency control diagram.

(10) Discuss briefly the problems that arise in the absence of concurrency control

(11) Resolve deadlock and starvation problems.

11 (12) Two phase locking system used in concurrency control

(13) Explain in brief transach schedule based on serializibilty and recoverability.

imp (14) which of the following schedules is (conflict) serializable? for each serializable schedule, determine the equivalent serial schedules.

$r1(x)$; $r3(x)$; $w1(x)$; $r2(x)$; $w3(x)$

$r1(x)$; $r3(x)$; $w3(x)$; $w1(x)$; $r2(x)$

$r3(x)$; $r2(x)$; $w3(x)$; $r1(x)$; $w1(x)$

$r3(x)$; $r2(x)$; $r1(x)$; $w3(x)$; $w1(x)$

(16) Explain why transaction should be atomic. Explain ACID properties. by considering the following transactions.

T1: read(A) A := A - 50 write(A) Read(B) B := B + 50 write(B)

DATABASE MANAGEMENT SYSTEM

(UNIT 5 AND UNIT 4 ANSWERS)

KRUTHIK B

1D19CS077

B SECTION

UNIT 5

5)

1. Users should be able to regard the execution of each transaction as atomic:

Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (say, when a system crash occurs).

2. Each transaction, run by itself with no concurrent execution of other transactions, lnust preserve the consistency of the database.

The DBMS assumes that consistency holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

3. Users should be able to understand a transaction without considering the effect of other concurrently executing transactions, even if the DBMS interleaves the actions of several transactions for performance reasons.

This property is sometimes referred to as isolation: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.

4. Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk.

This property is called durability.

The acronym ACID is sometimes used to refer to these four properties of transactions: atomicity, consistency, isolation and durability.

1)and 7)

In lock-based concurrency control, conflicting actions of different transactions are ordered by the order in which locks are obtained, and the lock protocol extends this ordering on actions to transactions, thereby ensuring serializability.

In optirrlistic concurrency control, a time stamp ordering is irnposed on transactions and validation checks that all conflicting actions occurred in the same order.

Time stamps can also be used in another way:

Each transaction can be assigned a timestamp at startup, and we can ensure, at execution tirne, that if action a(i) of transaction T(i) conflicts with action a(j) of transaction T(j),

a(i) occurs before a(j) if TS(T(i)) < TS(Tj). If an action violates this ordering, the transaction is aborted and restarted.

To implement this concurrency control schedule every database object 0 is givena read timestamp RTS(0) and a write timestamp WTS (0).

If transaction T wants to read object 0, and TS(T) < WTS(O), the order of this read with respect to the most recent write on 0 would violate the timestamp order between this transaction and the writer.

Therefore, T is aborted and restarted with a new, larger timestarnp. If TS(T) > WTS(O), Treads 0, and RTS(O) is set to the larger of RTS(O) and TS(T).

(Note that a physical change of the change to RTS(O) is written to disk and recorded in the log for recovery purposes, even on reads. This write operation is a significant overhead.)

Observe that if T is restarted with the same timestamp, it is guaranteed to be aborted again, due to the same conflict.

Contrast this behavior with the use of timestamps in 2PL for deadlock prevention, where transactions are restarted with the same timestarnp as before to avoid repeated restarts.

This shows that the two uses of timestamps are quite different and should not be confused.

2) and 11)

No Starvation concept for us only deadlock

Deadlocks

Consider the following example. Transaction *T1* sets an exclusive lock on object *A*, *T2* sets an exclusive lock on *B*, *T1* requests an exclusive lock on *B* and is queued, and *T2* requests an exclusive lock on *A* and is queued. Now, *T1* is waiting for *T2* to release its lock and *T2* is waiting for *T1* to release its lock. Such a cycle of transactions waiting for locks to be released is called a **deadlock.** Clearly, these two transactions will make no further progress. Worse, they hold locks that may be required by other transactions. The DBMS must either prevent or detect (and resolve) such deadlock situations; the common approach is to detect and resolve deadlocks.

Deadlocks Prevention

Elnpirical results indicate that deadlocks are relatively infrequent, and detection-based schemes work well in practice. However, if there is a high level of contention for locks and therefore an increased likelihood of deadlocks, prevention-based schelnes could perform better. We can prevent deadlocks by giving each transaction a priority and ensuring that lower-priority transactions are not allowed to wait for higher-priority transactions (or vice versa). One way to assign priorities is to give each transaction a timestamp when it starts up. The lower the timestamp, the higher is the transaction's priority; that is, the oldest transaction has the highest priority.

If a transaction *Ti* requests a lock and transaction ***Tj*** holds a conflicting lock, the lock lnanager can use one of the following two policies:

- Wait-die: If *Ti* has higher priority, it is allowed to wait; otherwise, it is aborted.
- Wound-wait: If *Ti* has higher priority, abort *Tj*; otherwise, *Ti* waits.

In the \vait-die scherne, lower-priority transactions can never wait for higher-priority transactions. In the wound-wait scherne, higher-priority transactions never wait for lower-priority transactions. In either ease, no deadlock cyc.le develops.

4)and12)

2 Phase Locking

The most widely used locking protocol, called *Strict Two-Phase Locking*, or *Strict 2PL*, has two rules. The first rule is

> 1. If a transaction T wants to *read* (respectively, *modify*) an object, it first requests a shared (respectively, exclusive) lock on the object.

Of course, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a transaction holds an exclusive lock on an object, no other transaction holds a shared or exclusive lock on the same object. The second rule in Strict 2PL is

> 2. All locks held by a transaction are released when the transaction is completed.

Requests to acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details. eWe discuss how application programmers can select properties of transactions and control locking overhead in Section 16.6.3.)

This is how Strict2PL guarantees Strict Schedule

A schedule is said to be strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or eOlnrnits. Strict schedules are recoverable, do not require cascading aborts, and actions of aborted transactions can be undone by restoring the original values of lnodified objects. (See the last exaInple in Section 16.3.4.) Strict 2PL irnproves on 2PL by guaranteeing that every allowed schedule is strict in addition to being conflict serializable. The reason is that when a transaction T writes an object under Strict 2PL, it holds the (exclusive) lock until it conunits or aborts. Thus, no other transaction can see or rnodify this object until T is cornplete.

3)

Multigranularity Locking

Another specialized locking strategy, called **multiple-granularity locking,** allows us to efficiently set locks on objects that contain other objects.

For instance, a database contains several files, a file is a collection of pages, and a page is a collection of records. A transaction that expects to access rnost of the pages in a file should probably set a lock on the entire file, rather than locking individual pages (or records) when it needs thern. Doing so reduces the locking overhead considerably. On the other hand, other tra,nsactions that require access to parts of the file—even parts not needed by this transaction—are blocked. If a transaction accesses relatively few pages of the file, it is better to lock only those pages. Sirnilarly, if a transaction accesses several records on a page, it should lock the entire page, and if it accesses just a few records, it should lock just those records.

The question to be addressed is how a lock rnanager can efficiently ensure that a page, for exaruple, is not locked by a transaction while another transaction holds a conflicting lock on the file containing the page (a.nd therefore, irnplicitly, on the page).

Explain Second para in your own words and do the rest

The idea is to exploit the hierarchical nature of the 'contains' relationship. A database contains a set of files, each file contains a set of pages, and each page contains a set of records. This containunent hierarchy can be thought of as a tree of objects, \vhere each node contains all its children. (The approach can easily be extended to cover hierarchies that are not trees, but we do not discuss this extension.) A lock on a node locks that node and, irnplicitly, all its descendants. (Note that this interpretation of a lock is very different fronl B+ tree locking, where locking a node does *not* lock any descendants ilnplicitly.)

In addition to shared (8) and exclusive *(X0)* locks, rnultiple-granularity locking protocols also use two new kinds of locks, called intention shared (18) and intention exclusive *(IX)* locks. 18 locks conflict only with X locks. IX locks conflict with 8 and X locks. To lock a node in *S* (respectively, X) luode, a transaction must first lock all its ancestors in 18 (respectively, 1*X)* rllode. Thus, if a transaction locks a node in 8 rIlode, no other transaction can have locked any ancestor in X rnode; siInilarly, if a transaction locks a node in X mode, no other transaction can have locked any ancestor in 8 or X mode. This ensures that no other transaction holds a lock on an ancestor that conflicts with the requested 8 or X lock on the node.

In this second para is a example read and understand it and then write

6)

A **schedule** is a list of actions (reading, writing, aborting, or committing) from a set of transactions, and the order in which two actions of a transaction *T* appear in a schedule must be the same as the order in which they appear in *T*. Intuitively, a schedule represents an actual or potential execution sequence. For example, the schedule in Figure 16.1 shows an execution order for actions of two transactions *T1* and *T2*. We move forward in time as we go down from one row to the next. We emphasize that a schedule describes the actions of transactions *as seen by the DBMS*. In addition to these actions, a transaction rnay carry out other actions, such as reading or writing from operating system files, evaluating arithmetic expressions, and so on; however, we assume that these actions do not affect other transactions; that is, the effect of a transaction on another transaction can be understood solely in terms of the cornmon database objects that they read and write.

T1	*T2*
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Figure 16.1 A Schedule Involving Two Transactions

Note that the schedule in Figure 16.1 does not contain an abort or commit action for either transaction. A schedule that contains either an abort or a commit for each transaction whose actions are listed in it is called a complete schedule. A complete schedule must contain all the actions of every transaction that appears in it. If the actions of different transactions are not interleaved-that is, transactions are executed from start to finish, one by one-we call the schedule a serial schedule.

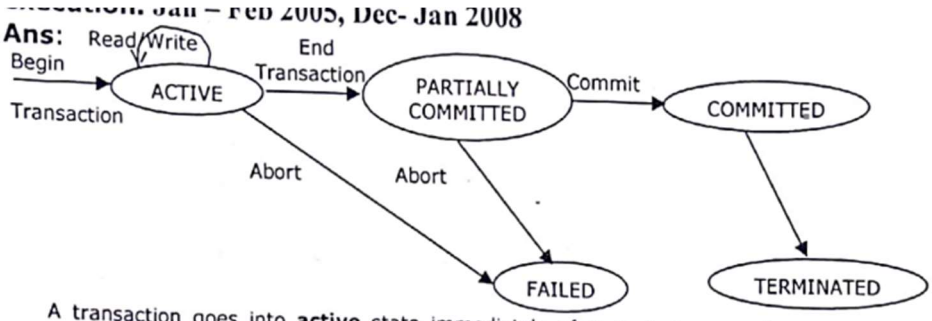
8)and13)

<u>Serializability(page no. 525 in TextBook) and Recoverability(page no. 571 in TexBook)</u>

Extra Imp Question

Imp) **Various States in Transaction**

Ans:



A transaction goes into **active** state immediately after it starts execution, where it can issue READ & WRITE operations. When the transaction ends, it moves to the **partially committed** state. At this point certain checks be made to ensure that the transaction did not interfere with execution of other transactions & a system failure will not result in an inability to record the changes of the transaction permanently. Once both checks are successful, the transaction is said to have reached its commit point & enters the **committed** state, which indicates the successful execution of that transaction.

A transaction can go to the **failed** state if one of checks fail or it is aborted during its active sate. The transaction may then have to be rolled back, to undo the effect of its write operations on the database. The **terminated** corresponds to the transaction leaving the system. Failed or aborted transactions may be restarted later as brand new transactions.

The various transaction states are as follows:

1. BEGIN TRANSACTION: This marks the beginning of transaction execution.
2. READ OR WRITE: These specify the read or write operations on the database items that are executed as part of a transaction.
3. END TRANSACTION: This specifies that read & write operations of a transaction have ended. At this poin, it may be necessary to check whether the changes done by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it has violated the concurrency control or for some other reason.
4. COMMIT TRANSACTION: This signals the successful end of the transaction so that any changes executed by the transaction can be safely committed to the database & will not be undone.
5. ROLLBACK (OR ABORT): This signals that the transaction has ended unsuccessful, so that any changes executed by the transaction may have applied to the database must be undone.

UNIT4

5) and 7)

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

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

a) Imparting Clear Semantics to Attributes in Relations

Group the attributes forming a relational schema with certain meaning associated with the attributes. The easier it is to explain the semantics of the relation, the better the relational schema design. Schemas should have well defined & unambiguous interpretations having clear semantics.

Guideline 1. Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

Examples of Guideline 1 Violation: Although there is nothing wrong logically with the relation, it violates Guideline 1 by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

b) Redundant Information in Tuples and Update Anomalies

One of the many purposes of schema design is to minimize storage space that the relation occupies. Grouping attributes into relational schema has a significant effect on storage space & we also have to deal with the problem of update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

i) Insertion Anomalies: Insertion anomalies can be differentiated into two types,

illustrated by the following examples based on the EMP_DEPT relation:

> Considering the relation EMP_DEPT(<u>SSN,</u> Ename, Bdate, Addr, Dno, Dname, Dmgrssn) & the relation EMP_PROJ(<u>SSN,Pno,</u> Hrs, Ename,Pname), it is difficult to insert a new department that has no employees. The only way to do this is to place NULL values in the attribute for employee.
>
> It is not possible to insert employee details for an employee who has not been assigned to any department. In such a case, we have to include NULL values.

ii) Deletion Anomalies: This anomaly occurs if we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is also lost inadvertently from the database.

iii) Modification Anomalies: In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all employees who work in that department; otherwise, the* database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

It is easy to see that these three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates that can be avoided; hence, we can state the next guideline as follows.

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

c) NULL Values in Tuples: If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes. Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons, if NULL values are present, the results may become unpredictable. NULLs can have multiple interpretations, such as the following:

- The attribute *does not apply to this tuple. For example, Visa_status may not* apply to U.S. students.
- The attribute value for this tuple is *unknown. For example, the Date_of_birth* may be unknown for an employee.
- The value is *known but absent; that is, it has not been recorded yet. For* example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

Guideline 3. As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

d) Disallowing the possibility of generating Spurious Tuples:

Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ relation.

EMP_LOCS

Ename	Plocation

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

Joining the above two relations using natural join, is going to result in generation of some extra spurious tuples which amounts to incorrect information. This is because, in this case, Plocation is the common attribute relating EMP_LOCS & EMP_PROJ1, which is neither a primary key nor a foreign key.

Guideline 4. Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

1),5) and 4)

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

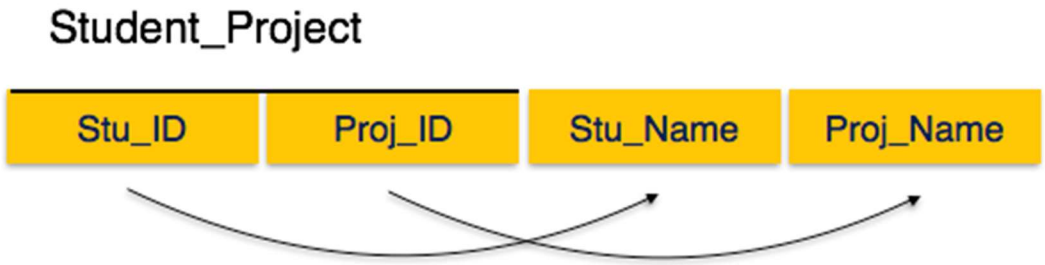
Each attribute must contain only a single value from its pre-defined domain.

Second Normal Form

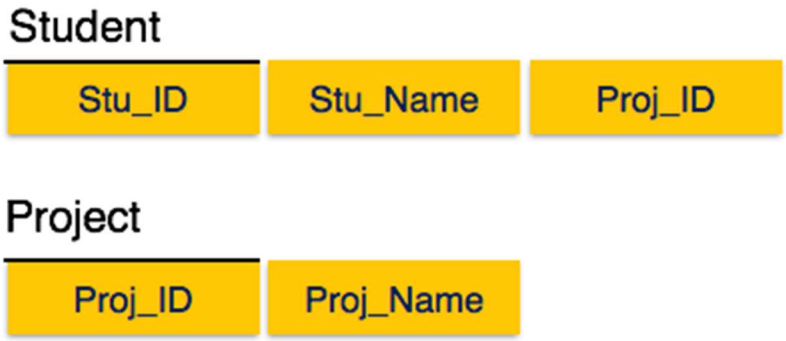
Before we learn about the second normal form, we need to understand the following −

- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

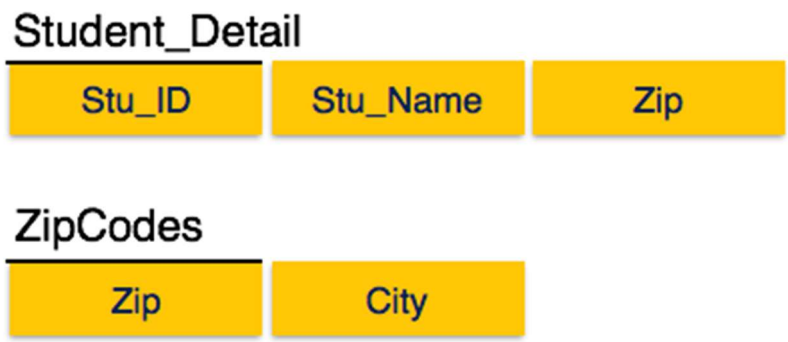
Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
    - X is a superkey or,
    - A is prime attribute.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows –



Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF.

Fourth normal form

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

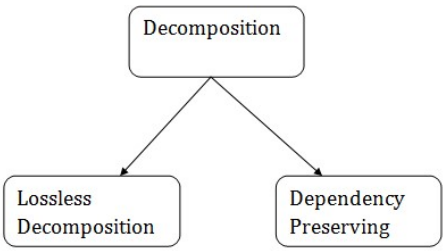
(For Fourth Question give and example of 3NF and show them that it is still not in BCNF as BCNF tell that the RHS must be a superkey and it should not be on the left side of any functional dependency)

6)

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

MULTIVALUED DEPENDENCY

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

```
BIKE_MODEL  →  →  MANUF_YEAR
BIKE_MODEL  →  →  COLOR
```

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

2,3,9 Questions shd be done will do before exam