

Q. No.	<p style="text-align: center;">VTU Exams March 2021 Subject: Database management Systems Code 18CS53 Solutions</p>	Mark s
1.a	<p>i) Database: A database is an organized collection of data, so that it can be easily accessed and managed. You can organize data into tables, rows, columns, and index it to make it easier to find relevant information.</p> <p>ii) DBMS Catalog: The database catalog of a database instance consists of metadata in which definitions of database objects such as base tables, views (virtual tables), synonyms, value ranges, indexes, users, and user groups are stored.</p> <p>iii) Entity: Entity in DBMS can be a real-world object with an existence, For example, in a College database, the entities can be Professor, Students, Courses, etc.</p> <p>iv) Snapshot: Database snapshots are like a view of a database as it was at a certain point in time. It is a read-only copy of the data and the state of the pages, which are made possible using a pointer file called the sparse file. Snapshot is a recent copy of the table from the database or a subset of rows/columns of a table. The SQL statement that creates and subsequently maintains a snapshot normally reads data from the database residing server. A snapshot is created on the destination system with the create snapshot SQL command. The remote table is immediately defined and populated from the master table.</p> <p>These are used to dynamically replicate data between distributed databases. Two types of snapshots are available.</p> <ol style="list-style-type: none"> 1. Simple snapshots 2. Complex snapshots <p>v) Degree of a relationship: The degree of a relationship is the number of entity types that participate (associate) in a relationship. By seeing an E-R diagram, we can simply tell the degree of a relationship i.e the number of an entity type that is connected to a relationship is the degree of that relationship.</p>	05
1.b	<p>Types of end users with example:</p> <p>Database users are categorized based up on their interaction with the data base.</p> <p>These are seven types of data base users in DBMS.</p> <ol style="list-style-type: none"> 1. Database Administrator (DBA) : Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database. The DBA will then create a new account id and password for the user if he/she need to access the data base. DBA is also responsible for providing security to the data base and he allows only the authorized users to access/modify the data base. 	05

	<ul style="list-style-type: none"> ○ DBA also monitors the recovery and back up and provide technical support. ○ The DBA has a DBA account in the DBMS which called a system or superuser account. ○ DBA repairs damage caused due to hardware and/or software failures. <p>2. Naive / Parametric End Users : Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results.</p> <p>For examples, Railway's ticket booking users are naive users. Clerk in any bank is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.</p> <p>3. System Analyst: System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.</p> <p>4. Sophisticated Users: Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own data base applications according to their requirement. They don't write the program code but they interact the data base by writing SQL queries directly through the query processor.</p> <p>5. Data Base Designers: Data Base Designers are the users who design the structure of data base which includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items to be related.</p> <p>6. Application Program: Application Program are the back end programmers who writes the code for the application programs. They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.</p> <p>7. Casual Users / Temporary Users : Casual Users are the users who occasionally use/access the data base but each time when they access the data base they require the new information, for example, Middle or higher level manager.</p>	
1.c.	<p>List of the advantages of using DBMS:</p> <p>In contrast with the File Based Data Management System, Dbms has numerous benefits. We are putting light on some of the considerable benefits here–</p> <p>1. Data Integrity</p>	10

Data integrity means data is consistent and accurate in the database. It is essential as there are multiple databases in DBMS. All these databases contain data which is visible to multiple users. Therefore, it is essential to ensure that data is consistent and correct in all databases for all users.

2. Data Security

Data security is a vital concept in a database. Only users authorized must be allowed to access the database and their identity must be authenticated using username and password. Unauthorized users shouldn't be allowed to access the database under any circumstances as it violets the integrity constraints.

A DBMS provides a better platform for data privacy thus helping companies to offer an improved data security.

3. Better data integration

Due to the database management system, we have access to well managed and synchronized form of data making it easy to handle. It also gives an integrated view of how a particular organization is working and keeps track of how one segment of the company affects another segment.

4. Minimized Data Inconsistency

Data inconsistency occurs between files when various versions of the same data appear in different places. Data consistency is ensured in the database; there is no data redundancy. Besides, any database changes are immediately reflected by all users, and there is no data inconsistency.

5. Faster Data Access

The database management system helps the users to produce quick answers to queries making data accessing accurate and faster. For any given dataset, dbms can help in solving insightful financial queries like:

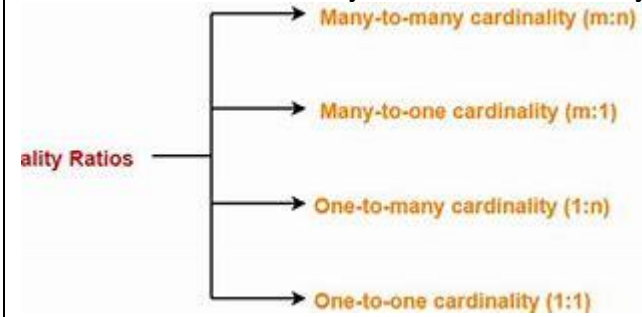
1. What is the bonus given to every salesperson in the last two months?

2. How many customers have a credit score or more than 800?

3. What is last year's profit?

4. Better decision making

Due to DBMS, we now have improved and managed data accessing because of which we can generate better quality information which can hence make better decisions.

	<p>Better quality ultimately improves validity, accuracy and time it takes to read data. It doesn't guarantee data quality; it provides a framework to make it easy to enhance data quality.</p> <p>5. Simplicity</p> <p>DBMS allows us to understand data better with a clear and simple logical view. With dbms, many operations like deletion, insertion or creation of file or data, are easy to implement.</p> <p>6. Recovery and Backup</p> <p>DBMS automatically takes care of recovery and backup. The users are not required to take periodical backup as this is taken care of by DBMS. Besides, it also restores a database after a system failure or crash to prevent its previous condition.</p> <p>7. Increased end-user productivity</p> <p>The available data transform into helpful information with the help of combination tools. It helps end users make better, informative and quick decisions that can make the difference between success and failure in the global economy.</p> <p>Additionally, today DBMS is also serving as the backbone of several advanced Technology practices like Data Science, Data Modeling and Machine Learning. So, if you are someone looking for a career in analytics or automation then DBMS is a must have skill for you.</p>	
<p>2.a</p>	<p>i) Cardinality: Cardinality is a relationship or joins between rows of 1 table to the rows of another table. A number that represents, one instance of first entity (table) is related to how many instances of the second entity, is known as cardinality.</p>  <p>ii) Weak Entity</p> <p><i>Weak entity</i></p> <ul style="list-style-type: none"> A weak entity is an entity set that does not have sufficient attributes for Unique Identification of its records 	<p>05</p>

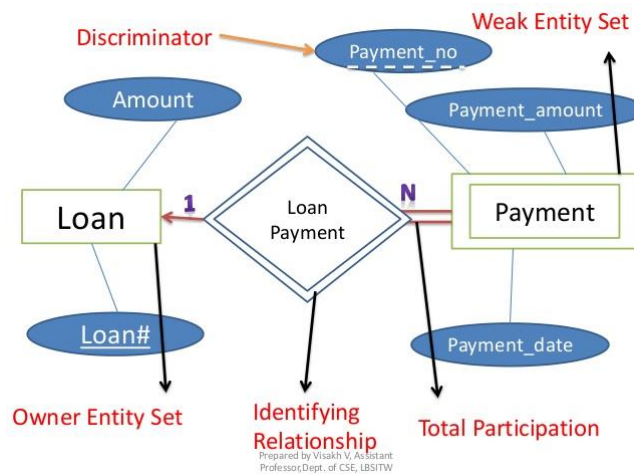
- Simply a weak entity is nothing but **an entity which does not have a primary key attribute**
- It **contains** a partial key called as **discriminator** which helps in identifying a **group of entities from the entity set**
- **Discriminator** is represented by underlining with a **dashed line**

Representation

- - A **double rectangle** is used for representing a **weak entity set**
 - The **double diamond** symbol is used for representing the **relationship between a strong entity and weak entity** which is known as identifying relationship
 - **Double lines** are used for presenting the **connection with a weak entity set** with relationship

Example for weak entity

- - In the ER diagram, we have **two entities building and apartment**
 - **Building is a strong entity** because it has a **primary key** attribute called building number which is capable of uniquely identifying all the flats present in the apartment
 - Unlike building, **apartment is weak entity** because it **does not have any primary key and door number here acts only as a discriminator** because door number cannot be used as a primary key, there might be multiple flats in the building with the same door number or on different floors.

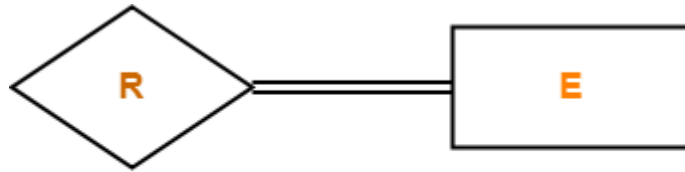


iii) Program data Independence

Data independence is the type of data transparency that matters for a centralized DBMS. It refers to the immunity of user applications to changes made in the definition and organization of data. Application programs should not, ideally, be exposed to details of data representation and storage. The DBMS provides an abstract view of the data that hides such details.

iv) 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

v) Value Sets:

A **value set** is a uniquely identifiable **set** of valid concept representations, where any concept representation can be tested to determine whether or not it is a member of the **value set**. A **value set** is typically used to represent the possible **values** of a coded data element in an information model.

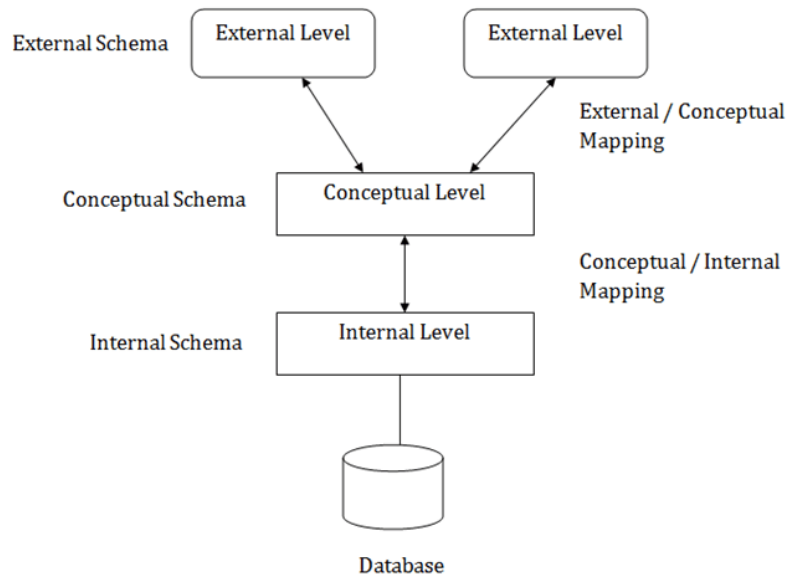
2.b

Three Schema Architecture:

05

The three-schema architecture is as follows:

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database.
- The three schema architecture contains three-levels. It breaks the database down into three different categories.



	<p>Mapping between Schema levels:</p> <p>1. Conceptual/Internal Mapping:</p> <ul style="list-style-type: none"> • The conceptual/internal mapping defines the correspondence between the conceptual view and the store database. • It specifies how conceptual record and fields are represented at the internal level. • It relates conceptual schema with internal schema. • If structure of the store database is changed. • If changed is made to the storage structure definition-then the conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant. • There could be one mapping between conceptual and internal levels. <p>2. External/Conceptual Mapping:</p> <ul style="list-style-type: none"> • The external/conceptual mapping defines the correspondence between a particular external view and conceptual view. • It relates each external schema with conceptual schema. • The differences that can exist between these two levels are analogous to those that can exist between the conceptual view and the stored database. • Example: fields can have different data types; fields and record name can be changed; several conceptual fields can be combined into a single external field. • Any number of external views can exist at the same time; any number of users can share a given external view: different external views can overlap. • There could be several mapping between external and conceptual levels. 	
2.c	<p>Different types of attributes in ER Model:</p> <p>1. Simple attribute: An attribute which cannot be further subdivided into components is a simple attribute.</p> <p><i>Example:</i> The roll number of a student, the id number of an employee.</p> <p>2. Composite attribute: An attribute which can be splitted into components is a composite attribute.</p> <p><i>Example:</i> The address can be further splitted into house number, street number, city, state, country and pincode, the name can also be splitted into first name middle name and last name.</p>	10

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

Example: The age of a student.

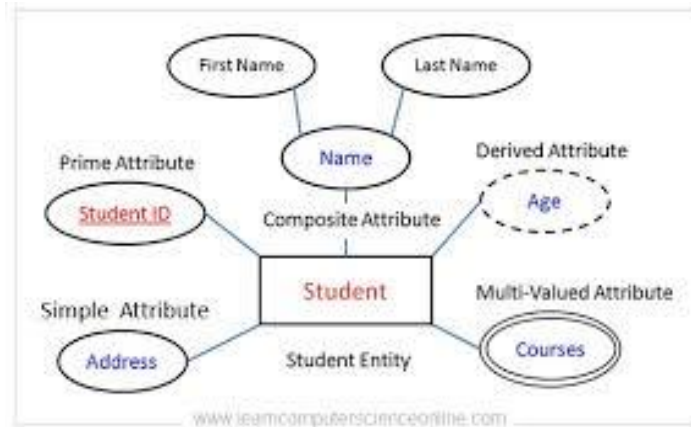
4. **Multi-valued attribute:**
The attribute which takes up more than a single value for each entity instance is multi-valued attribute.

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

5. **Derived attribute:**
An attribute that can be derived from other attributes is derived attribute.

Example: Total and average marks of a student.

Examples:



3.a **Entity Integrity constraint:**
The **entity integrity constraint** states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows. A table can contain a null value other than the primary key field.

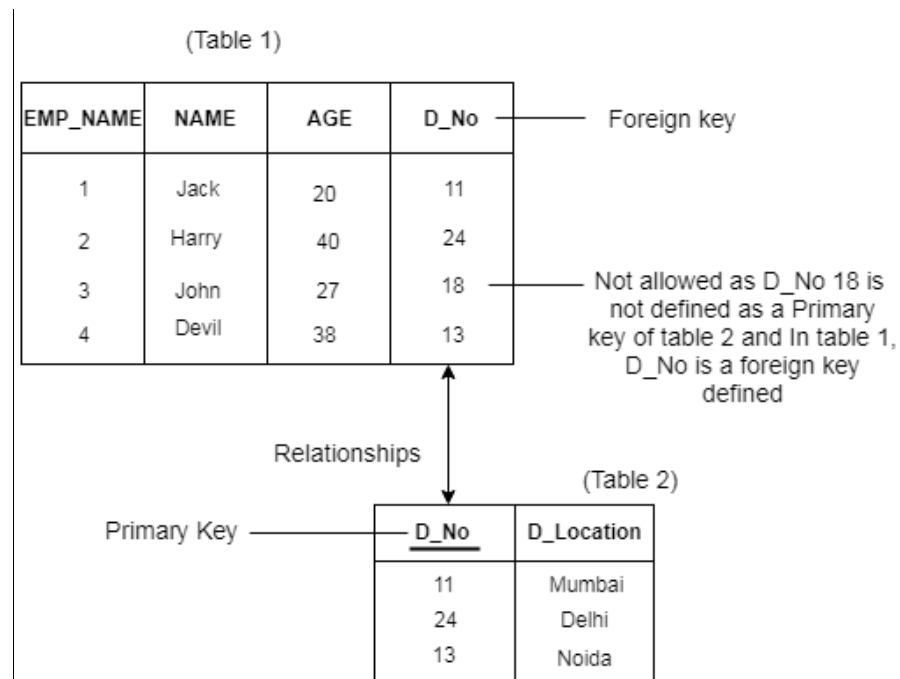
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Referential Integrity Constraint:

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



They are important because of-

They are important to enforce business constraints, Integrity and control redundancy.

	<p>Equijoin(\bowtie): Equijoin is a special case of conditional join where only equality condition holds between a pair of attributes. As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.</p> <p>Example: Select students whose ROLL_NO is equal to EMP_NO of employees</p> <p style="text-align: center;">STUDENT\bowtieSTUDENT.ROLL_NO=EMPLOYEE.EMP_NOEMPLOYEE</p> <p>Natural Join(\bowtie): It is a special case of equijoin in which equality condition hold on all attributes which have same name in relations R and S (relations on which join operation is applied). While applying natural join on two relations, there is no need to write equality condition explicitly. Natural Join will also return the similar attributes only once as their value will be same in resulting relation.</p> <p>Example: Select students whose ROLL_NO is equal to ROLL_NO of STUDENT_SPORTS as:</p> <p style="text-align: center;">STUDENT\bowtieSTUDENT_SPORTS</p>	
<p>3.c</p>	<p>Given the schema :</p> <p>Passenger (pid, pname, pgender, pcity)</p> <p>Agency (aid, anme, acity)</p> <p>Flight (fid, fdate, time, src, ddest)</p> <p>Booking (pid, aid, fid, fdate)</p> <p>Give relation algebra expression for the following :</p> <ol style="list-style-type: none"> Get the complete details of all flights to new Delhi Find only the flight numbers for passenger with paid 123 for flights to Chennai before 06/11/2020 Find the passenger names for those who do not have any bookings in any flights Get the details of flights that are scheduled on both dates 01/12/2020 and 02/12/2020 at 16:00 hours Find the details of all male passengers who are associated with jet agency. (10 Marks) <p>Ans</p> <p>(i) σ destination = "New Delhi" (flight)</p> <p>ii) σ src = "Chennai" ^ dest = "New Delhi" (flight)</p> <p>iii) Π fid (σ pid = 123 (booking) \bowtie σ dest = "Chennai" ^ fdate < 06/11/2020 (flight))</p> <p>iv) (σ fdate = 01/12/2020 ^ time = 16:00 (flight)) \cap (σ fdate = 02/12/2020 ^ time = 16:00 (flight))</p> <p>v) Π passengers.pid, pname, pcity (σ pgender = "Male" (passengers \bowtie booking \bowtie agency))</p>	<p>10</p>
<p>4a.</p>	<p>ER to relational mapping algorithm with:</p>	<p>10</p>

Here is the various cases that explain how to convert ER diagram to Relational Model for different scenarios.

Step 1: Mapping of Regular Entity Types.

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key for R.
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.
- **Example:** We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
 - SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The primary key of R is the *combination* of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
- **Example:** Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.
 - Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
 - The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

Step 3: Mapping of Binary 1:1 Relation Types

- 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.
- There are three possible approaches:
 - **Foreign Key approach:** Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.
 - **Example:** 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the

role of S, because its participation in the MANAGES relationship type is total.

- **Merged relation option:** An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
- **Cross-reference or relationship relation option:** The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

Step 4: Mapping of Binary 1:N Relationship Types.

- For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
- Include any simple attributes of the 1:N relation type as attributes of S.
- Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure.
 - For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

■ Step 5: Mapping of Binary M:N Relationship Types.

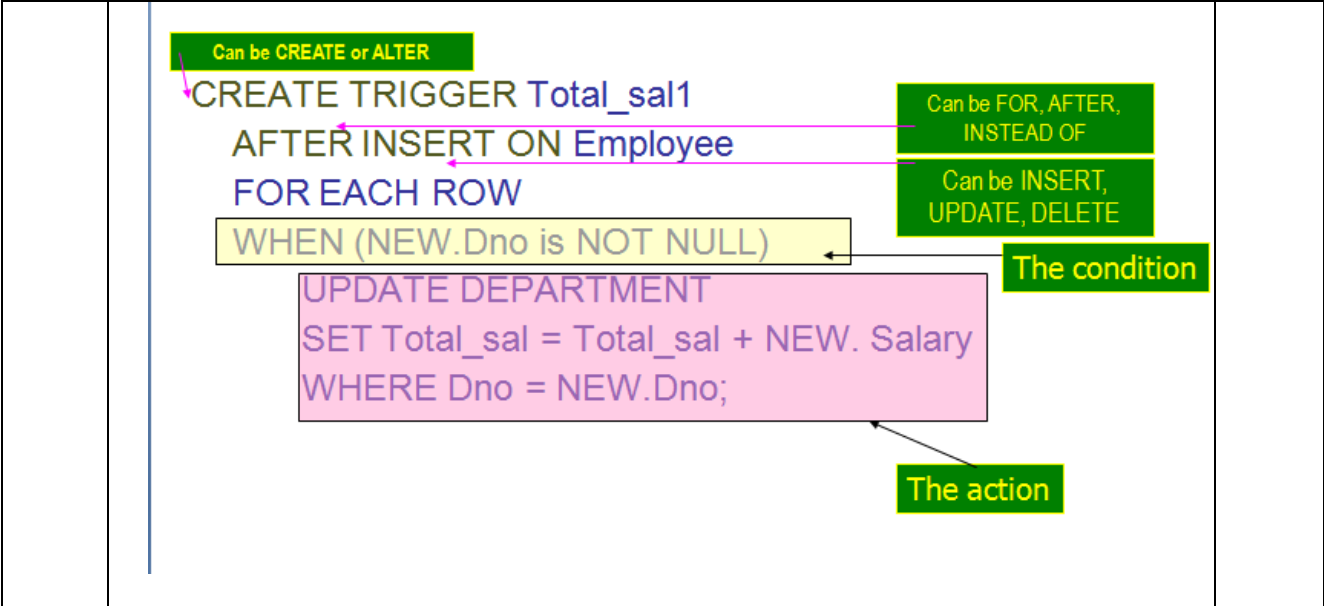
- For each regular binary M:N relationship type R, *create a new relation S to represent R.*
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; *their combination will form the primary key of S.*
- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
- Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema.
 - The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
 - Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

Step 7: Mapping of N-ary Relationship Types.

	<ul style="list-style-type: none"> ■ For each n-ary relationship type R, where $n > 2$, create a new relationship S to represent R. ■ Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. ■ Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S. ■ Example: The relationship type SUPPY in the ER on the next slide. <ul style="list-style-type: none"> ■ This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME} <p>Step8: Options for Mapping Specialization or Generalization.</p> <ul style="list-style-type: none"> ■ Convert each specialization with m subclasses $\{S_1, S_2, \dots, S_m\}$ and generalized superclass C, where the attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key, into relational schemas using one of the four following options: <ul style="list-style-type: none"> ■ Option 8A: Multiple relations-Superclass and subclasses ■ Option 8B: Multiple relations-Subclass relations only ■ Option 8C: Single relation with one type attribute ■ Option 8D: Single relation with multiple type attributes 	
<p>4.b</p>	<p>Write SQL query for the following database scheme :</p> <p>Employee(employee_name, street, city) Works (employee_name, company_name, salary) Company(company_name, city) Manages(employee_name, manager_name)</p> <ol style="list-style-type: none"> i) Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than \$10,000 ii) Find the names of all employees in the database who do not work for 'First Bank Corporation'. Assume that all people work for exactly one company iii) Find the names of all employees in the database who earn more than every employee of 'Small Bank Corporation'. Assume that all people work for at most one company iv) Find the name of the company that has the smallest payroll v) Find the names of all employees in the database who live in the same cities and on the same streets as do their managers. <p style="text-align: right;">(10 Marks)</p> <p>a. Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than \$10,000.</p> <p><i>select employee.employee-name, employee.street, employee.city from employee, works where employee.employee-name=works.employee-name and company-name = 'First Bank Corporation' and salary > 10000)</i></p> <p>b. Find the names of all employees in the database who do not work for 'First Bank Corporation'. Assume that all people work for exactly one company. <i>Select employee-name from works where company-name not in('First Bank Corporation');</i></p>	<p>10</p>

	<p>c. Find the names of all employees in the database who earn more than every employee of 'Small Bank Corporation'. Assume that all people work for at most one company. select employee-name from works where salary > all (select salary from works where company-name = 'Small Bank Corporation')</p> <p>d. Find the name of the company that has the smallest payroll Select C.comany_name, min(salary) as “smallest payroll” from Employee E, Works W, Company C where E.employee_name = W.employee_name and W.company_name=C.company_name Group by C.company_name ;</p> <p>e. Find the names of all employees in the database who live in the same cities and on the same streets as do their managers. select p.employee-name from employee p, employee r, manages m where p.employee-name = m.employee-name and m.manager-name = r.employee-name and p.street = r.street and p.city = r.city</p>	
<p>5.a</p>	<p>Explain the Cursors and its properties in Embedded SQL with example:</p> <p>A cursor is used to retrieve rows from a query that has multiple rows in its result set. A cursor is a handle or an identifier for the SQL query and a position within the result set.</p> <p>To manage the cursor in embedded SQL following steps to consider-</p> <ol style="list-style-type: none"> 1. Declare a cursor for a particular SELECT statement, using the DECLARE statement. 2. Open the cursor using the OPEN statement. 3. Retrieve results one row at a time from the cursor using the FETCH statement. 4. Fetch rows until the Row Not Found warning is returned. <p>Errors and warnings are returned in the SQLCA structure. See The SQL Communication Area (SQLCA).</p> <ol style="list-style-type: none"> 5. Close the cursor, using the CLOSE statement. <p>Example:</p> <pre>void print_employees(void) { EXEC SQL BEGIN DECLARE SECTION; char name[50]; char sex; char birthdate[15]; a_sql_len ind_birthdate; EXEC SQL END DECLARE SECTION; EXEC SQL DECLARE C1 CURSOR FOR SELECT GivenName ' ' Surname,</pre>	<p>05</p>

	<pre> Sex, BirthDate FROM Employees; EXEC SQL OPEN C1; for(;;) { EXEC SQL FETCH C1 INTO :name, :sex, :birthdate:ind_birthdate; if(SQLCODE == SQLE_NOTFOUND) { break; } else if(SQLCODE < 0) { break; } if(ind_birthdate < 0) { strcpy(birthdate, "UNKNOWN"); } printf("Name: %s Sex: %c Birthdate: %s.n",name, sex, birthdate); } EXEC SQL CLOSE C1; } </pre>	
<p>5.b</p>	<p>Trigger defined in these ways- Generalized Model for Active Databases and Oracle Triggers Triggers are executed when a specified condition occurs during insert/delete/update Triggers are action that fire automatically based on these conditions</p> <p>Generalized Model (contd.) Triggers follow an Event-condition-action (ECA) model Event: Database modification E.g., insert, delete, update), Condition: Any true/false expression Optional: If no condition is specified then condition is always true Action: Sequence of SQL statements that will be automatically executed When a new employees is added to a department, modify the Total_sal of the Department to include the new employees salary Logically this means that we will CREATE a TRIGGER, let us call the trigger Total_sal1 This trigger will execute AFTER INSERT ON Employee table It will do the following FOR EACH ROW WHEN NEW.Dno is NOT NULL The trigger will UPDATE DEPARTMENT By SETting the new Total_sal to be the sum of old Total_sal and NEW. Salary WHERE the Dno matches the NEW.Dno;</p>	<p>05</p>



<p>5.c</p>	<p>Insert, update, delete, update, alter and drop in SQL with examples:</p> <p>Create Table</p> <p>The CREATE TABLE statement is used to create a new table in a database. In that table, if you want to add multiple columns, use the below syntax.</p> <p>Syntax</p> <ol style="list-style-type: none"> 1. CREATE TABLE table_name (2. column1 datatype, 3. column2 datatype, 4. column3 datatype, 5. 6.); <p>The column parameters specify the names of the columns of the table.</p> <p>The data type parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).</p> <p>Create Table Example</p> <ol style="list-style-type: none"> 1. CREATE TABLE Employee(2. EmpId int, 3. LastName varchar(255), 4. FirstName varchar(255), 5. Address varchar(255), 6. City varchar(255) 	<p>10</p>
------------	---	-----------

7.);

The EmpId column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters and the maximum length for these fields is 255 characters.

Insert Value in this Table

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

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

Syntax

The first way specifies both the column names and the values to be inserted.

If you are adding values for all the columns of the table, then no need to specify the column names in the SQL query. However, make sure that the order of the values is in the same order as the columns in the table.

1. INSERT INTO table_name (column1, column2, column3, ...)
2. VALUES (value1, value2, value3, ...);
- 3.
4. '2nd way
5. INSERT INTO table_name
6. VALUES (value1, value2, value3, ...);

Example

Insert value in a 1st way. The column names are used here

1. INSERT INTO Employee (EmpId,LastName,FirstName,ADDRESS,City)
2. VALUES (1, 'XYZ', 'ABC', 'India', 'Mumbai');
3. INSERT INTO Employee (EmpId,LastName,FirstName,ADDRESS,City)
4. VALUES (2, 'X', 'A', 'India', 'Pune');

Insert value in a 2nd way.

1. INSERT INTO Employee
2. VALUES (3, 'XYZ', 'ABC', 'India', 'Mumbai');

Select Statment in SQL

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

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

1. SELECT column1, column2, ...
2. FROM table_name;

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

1. SELECT * FROM table_name;

If the above query is executed, then all record is displayed.

Example

1. Select EmpId, LastName from Employee;
2. Select * from Employee;

Update Table

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

Syntax

1. UPDATE table_name
2. SET column1 = value1, column2 = value2, ...
3. WHERE condition;

Example

1. UPDATE Employee
2. SET FirstName= 'KS', City= 'Pune'
3. WHERE EmpId= 1;

If the above query is executed then for EmpId= 1, "Firstname" and "City" column data will be updated.

Update Multiple Rows

It is the WHERE clause that determines how many records will be updated.

	<p>1. UPDATE Employee 2. SET City='Pune'</p> <p>Delete Statment in SQL</p> <p>The DELETE statement is used to delete existing records in a table for a particular Record.</p> <p>Syntax</p> <p>1. DELETE FROM table_name WHERE condition;</p> <p>Example</p> <p>1. DELETE FROM Employee WHERE EmpId=1;</p> <p>In Employee table EmpId = 1 record gets deleted.</p> <p>Alter Statement in SQL</p> <p>Alter command is used to change the physical structure of a table such as adding or removing columns, changing their data types, adding or removing business rules etc.</p> <p>Example</p> <p>Alter table Employee add Mobile varchar(12);</p> <p>Drop Statement in SQL</p> <p>This command is used to delete a table along with their structure.</p> <p>Example</p> <p>Drop table Employee;</p>	
<p>6.a</p>	<p>Stored Procedure in SQL: They are the procedure stored after compilation in database and can be executed many times as per the use. Syntax: □ DELIMITER &&</p>	<p>05</p>

	<p> <input type="checkbox"/> CREATE PROCEDURE procedure_name [[IN OUT INOUT] parameter_name datatype [, parameter datatype]] <input type="checkbox"/> BEGIN <input type="checkbox"/> Declaration_section <input type="checkbox"/> Executable_section <input type="checkbox"/> END && <input type="checkbox"/> DELIMITER ; </p> <p>IN parameter</p> <p>It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.</p> <p>OUT parameters</p> <p>It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.</p> <p>INOUT parameters</p> <p>It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.</p> <p>CALL procedure_name (parameter(s)) Example:</p> <ol style="list-style-type: none"> 1. DELIMITER && 2. CREATE PROCEDURE get_merit_student () 3. BEGIN 4. SELECT * FROM student_info WHERE marks > 70; 5. SELECT COUNT(stud_code) AS Total_Student FROM student_info; 6. END && 7. DELIMITER ; 	
<p>6.b</p>	<p>Types of JDBC drivers:</p> <p>Today, there are five types of JDBC drivers in use:</p> <ul style="list-style-type: none"> • Type 1: JDBC-ODBC bridge • Type 2: partial Java driver 	<p>05</p>

- **Type 3:** pure Java driver for database middleware
- **Type 4:** pure Java driver for direct-to-database
- **Type 5:** highly-functional drivers with superior performance

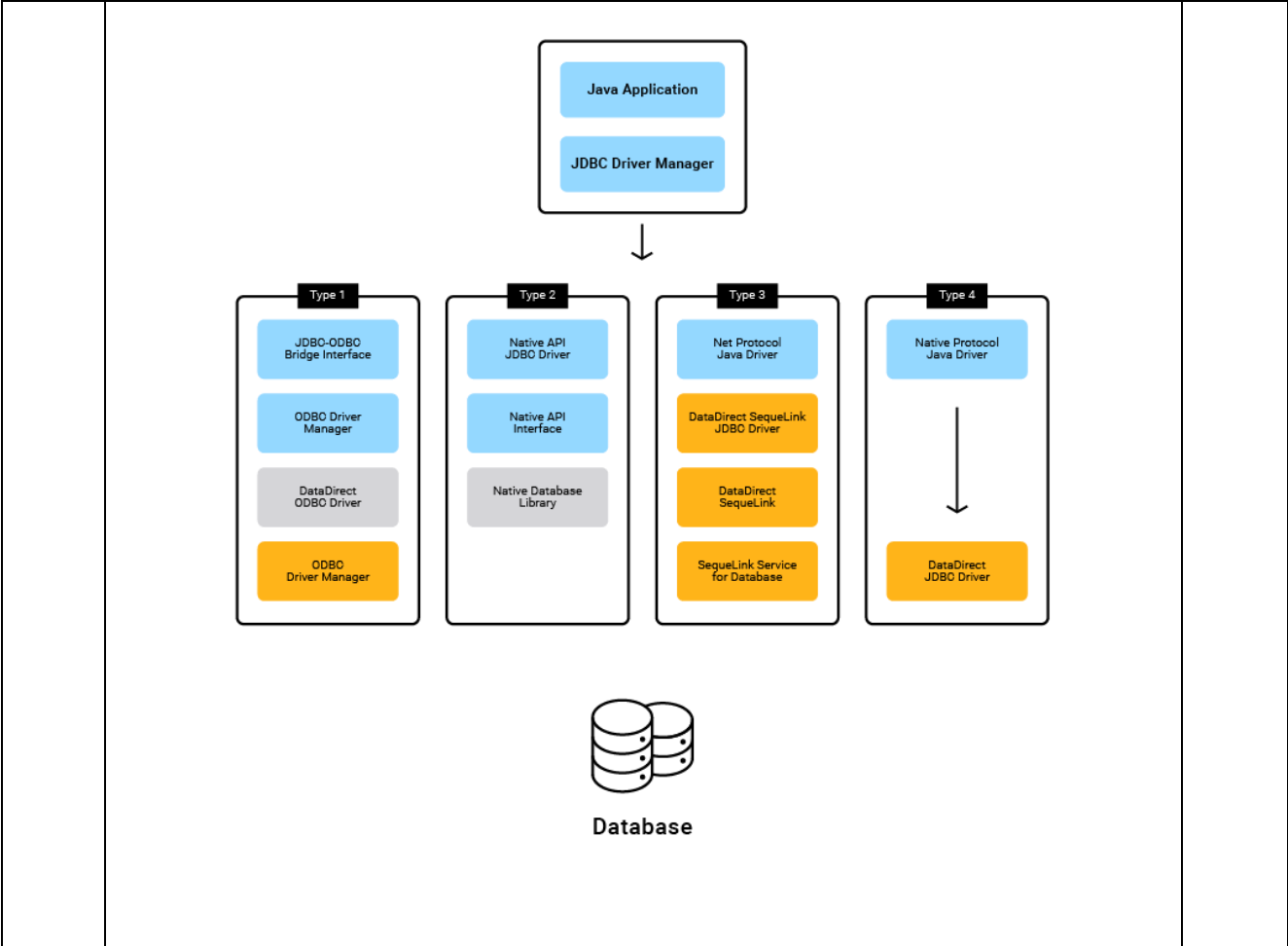
Type 5 JDBC drivers (such as [DataDirect JDBC drivers](#)) offer advanced functionality and superior performance over other driver types.

Type 4 drivers are the most common and are designed for a particular vendor's database.

In contrast, **Type 3 is a single JDBC driver** used to access a middleware server, which, in turn, makes the relevant calls to the database. A good example of Type 3 JDBC driver is the [DataDirect SequeLink JDBC](#) driver.

Type 1 JDBC drivers are used for testing JDBC applications against an ODBC data source. **Type 2 JDBC drivers** require a native database API to be used. Both Type 1 and Type 2 JDBC driver types mix a Java-based API with another API.

The following figure shows a side-by-side comparison of the implementation of each of the JDBC driver types. All four implementations show a Java application or applet using the JDBC API to communicate through the JDBC Driver Manager with a specific JDBC driver type.



6.c

Aggregate functions in SQL:

10

Aggregate Function	Descriptions
count()	It returns the number of rows, including rows with NULL values in a group.
sum()	It returns the total summed values (Non-NULL) in a set.
average()	It returns the average value of an expression.
min()	It returns the minimum (lowest) value in a set.
max()	It returns the maximum (highest) value in a set.
group_concat()	It returns a concatenated string.
first()	It returns the first value of an expression.
last()	It returns the last value of an expression.

Count() Function

MySQL count() function **returns the total number of values** in the expression. This function produces all rows or only some rows of the table based on a specified condition, and its return type is **BIGINT**. It returns zero if it does not find any matching rows. It can work with both numeric and non-numeric data types.

1. mysql> SELECT SUM(working_hours) AS "Total working hours" FROM employee;

AVG() Function

MySQL AVG() function **calculates the average of the values** specified in the column. Similar to the SUM() function, it also works with numeric data type only.

Suppose we want to get the average working hours of all employees in the table, we need to use the AVG() function as shown in the following query:

```
mysql> SELECT AVG(working_hours) AS "Average working hours" FROM employee;
```

MIN() Function

MySQL MIN() function **returns the minimum (lowest) value** of the specified column. It also works with numeric data type only.

Suppose we want to get minimum working hours of an employee available in the table, we need to use the MIN() function as shown in the following query:

1. mysql> SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;

MAX() Function

MySQL MAX() function **returns the maximum (highest) value** of the specified column. It also works with numeric data type only.

Suppose we want to get maximum working hours of an employee available in the table, we need to use the MAX() function as shown in the following query:

```
mysql> SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;
```

LAST() Function

	<p>This function returns the last value of the specified column. To get the last value of the column, we must have to use the ORDER BY and LIMIT clause. It is because the LAST() function only supports in MS Access.</p> <p>1. mysql> SELECT working_hours FROM employee ORDER BY name DESC LIMIT 1;</p> <p>GROUP_CONCAT() Function</p> <p>The GROUP_CONCAT() function returns the concatenated string from multiple rows into a single string. If the group contains at least one non-null value, it always returns a string value. Otherwise, we will get a null value.</p> <p>Etc.</p>	
<p>7.a</p>	<p>Types of update anomalies with example:</p> <p>Anomalies</p> <p>1- Update Anomaly: Let say we have 10 columns in a table out of which 2 are called employee Name and employee address. Now if one employee changes it's location then we would have to update the table. But the problem is, if the table is not normalized one employee can have multiple entries and while updating all of those entries one of them might get missed.</p> <p>2- Insertion Anomaly: Let's say we have a table that has 4 columns. Student ID, Student Name, Student Address and Student Grades. Now when a new student enroll in school, even though first three attributes can be filled but 4th attribute will have NULL value because he doesn't have any marks yet.</p> <p>3- Deletion Anomaly: This anomaly indicates unnecessary deletion of important information from the table. Let's say we have student's information and courses they have taken as follows (student ID, Student Name, Course, address). If any student leaves the school then the entry related to that student will be deleted. However, that deletion will also delete the course information even though course depends upon the school and not the student.</p>	<p>05</p>
<p>7.b</p>	<p>Armstrong Inference rules:</p> <ul style="list-style-type: none"> ■ Armstrong's inference rules: <ul style="list-style-type: none"> ■ IR1. (Reflexive) If Y <i>subset-of</i> X, then X -> Y ■ IR2. (Augmentation) If X -> Y, then XZ -> YZ <ul style="list-style-type: none"> ■ (Notation: XZ stands for X U Z) ■ IR3. (Transitive) If X -> Y and Y -> Z, then X -> Z ■ IR1, IR2, IR3 form a sound and complete set of inference rules <ul style="list-style-type: none"> ■ These are rules hold and all other rules that hold can be deduced from these 	<p>05</p>

- Some additional inference rules that are useful:
 - **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
 - **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
 - **Pseudotransitivity:** If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

7.c

Need for Normalization:

Normalization is a technique for organizing data in a database. It is important that a database is **normalized** to minimize redundancy (duplicate data) and to ensure only related data is stored in each table. It also prevents any issues stemming from database modifications such as insertions, deletions, and updates.

10

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

Normal Forms with Example:

1NF
2NF
3NF

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.

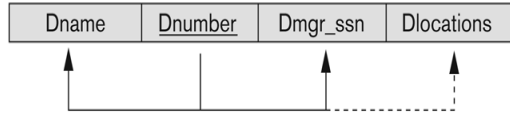
1NF:

Disallows

- composite attributes
- multivalued attributes
- **nested relations;** attributes whose values for an *individual tuple* are non-atomic
- Considered to be part of the definition of relation

(a)

DEPARTMENT



(b)

DEPARTMENT

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

(c)

DEPARTMENT

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

2NF:

- A relation schema **R** is in second normal form (2NF) if every non-prime attribute **A** in **R** is fully functionally dependent on the primary key
- **R** can be decomposed into 2NF relations via the process of 2NF normalization

Figure 10.8

Normalization into 1NF.
 (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

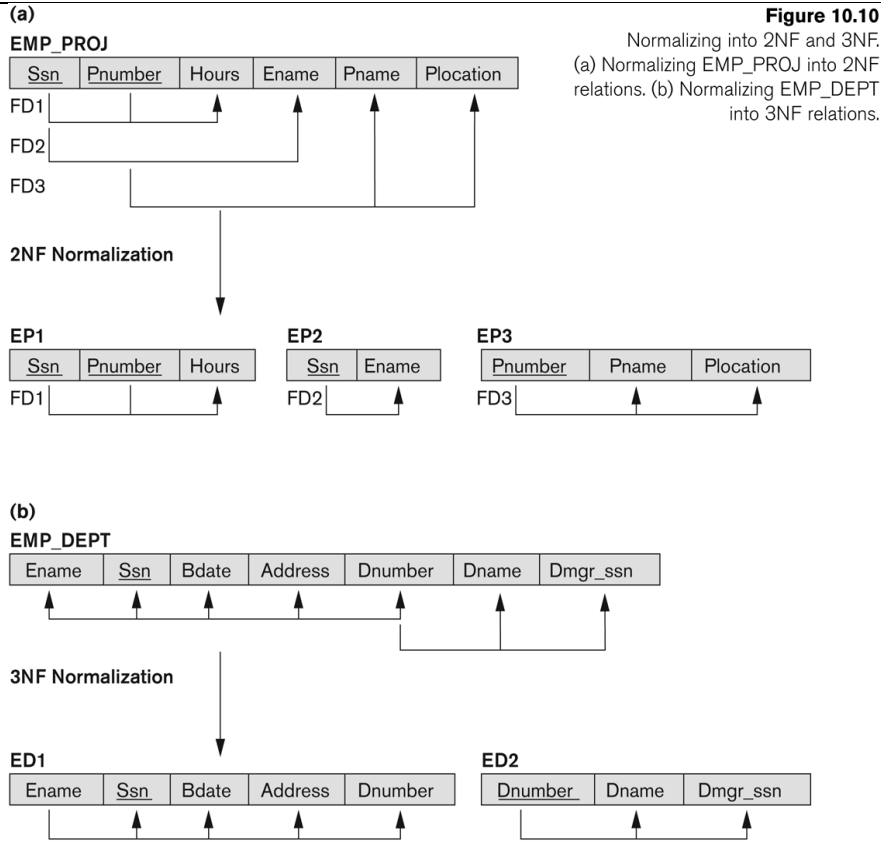


Figure 10.10

Normalizing into 2NF and 3NF.
 (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

3NF:

- **Definition:**
 - **Transitive functional dependency:** a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$
- **Examples:**
 - $SSN \rightarrow DMGRSSN$ is a transitive FD
 - Since $SSN \rightarrow DNUMBER$ and $DNUMBER \rightarrow DMGRSSN$ hold
 - $SSN \rightarrow ENAME$ is non-transitive
 - Since there is no set of attributes X where $SSN \rightarrow X$ and $X \rightarrow ENAME$
- A relation schema R is in third normal form (3NF) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- **NOTE:**
 - In $X \rightarrow Y$ and $Y \rightarrow Z$, with X as the primary key, we consider this a problem only if Y is not a candidate key.
 - When Y is a candidate key, there is no problem with the transitive dependency .
 - E.g., Consider EMP (SSN, Emp#, Salary).
 - Here, $SSN \rightarrow Emp\# \rightarrow Salary$ and $Emp\#$ is a candidate key.

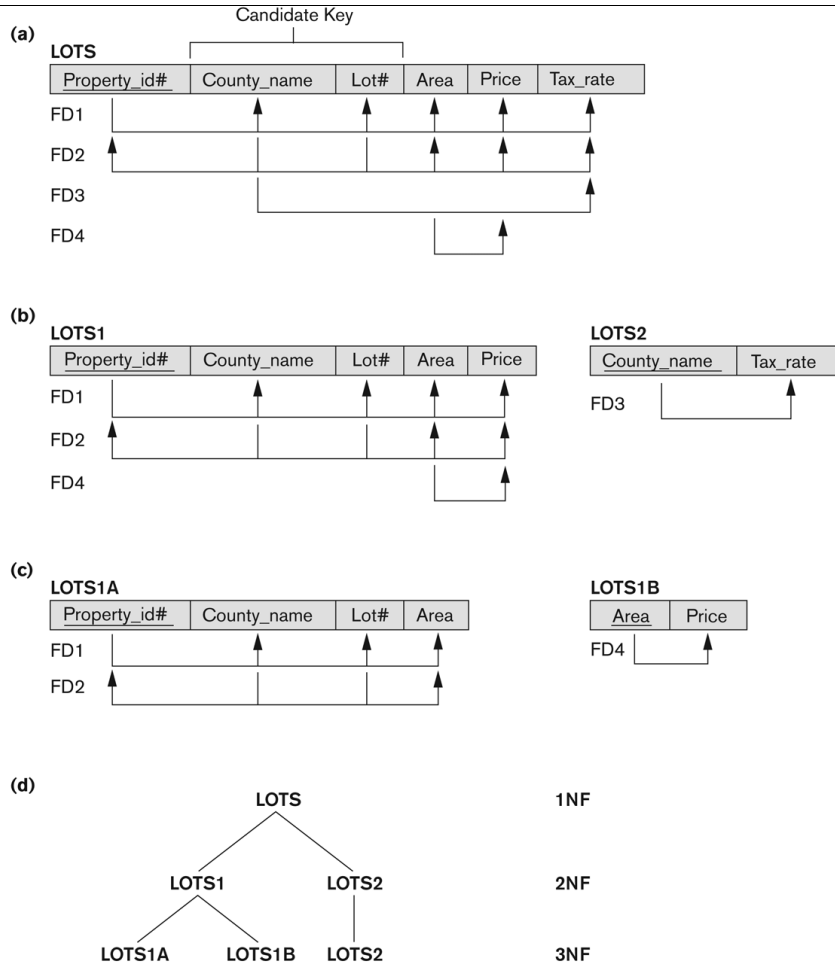


Figure 10.11 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

8.a

What is functional dependency? Write an algorithm to find minimal cover for set of functional dependencies. Construct minimal cover m for set of functional dependencies which are : $E : \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$ (10 Marks)

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Trivial functional dependency

- $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Non-trivial functional dependency

- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then $A \rightarrow B$ is called as complete non-trivial.

8.b Consider the schema $R = ABCD$, subjected to FDs $F = \{A \rightarrow B, B \rightarrow C\}$, and the non-binary partition $D1 = \{ACD, AB, BC\}$. State whether D1 is a lossless decomposition? [give all steps in detail]. (10 Marks)

R1={ ACD}
R2={AB}
R3={BC}

$R1 \cup R2 \cup R3 = R \Rightarrow \{ACD\} \cup \{AB\} \cup \{BC\} = \{ABCD\} = R$

	A	B	C	D
R1	*		*	*
R2	*	*		
R3		*	*	

R1 Join R2= $\Rightarrow \{ABCD\}$

A is common Attribute
 $A^+ = ABC$
A is not a candidate key

R2 Join R3= $\Rightarrow \{ABC\}$

B is a common attribute
 $B^+ = BC$
B is not a Candidate Key

All the columns are covered while joining the relations
But Based on Functional dependency does not any candidate key it is not a lossless join

9.a **Transaction:** 05

ACID Properties:

	<p>ACID properties</p> <ul style="list-style-type: none"> • Atomicity A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all. • Consistency preservation A transaction is consistency preserving if its complete execution takes the database from one consistent state to another • Isolation The execution of a transaction should not be interfered with by any other transactions executing concurrently • Durability 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 	
--	---	--

<p>9.b</p>	<p>Transition diagram of a transaction with explanation:</p> <p>Transaction states</p> <ul style="list-style-type: none"> • BEGIN_TRANSACTION: marks start of transaction • READ or WRITE: two possible operations on the data • END_TRANSACTION: marks the end of the read or write operations; start checking whether everything went according to plan • COMMIT_TRANSACTION: signals successful end of transaction; changes can be “committed” to DB • Partially committed • ROLLBACK (or ABORT): signals unsuccessful end of transaction, changes applied to DB must be undone <pre> graph LR Start(()) -- BEGIN TRANSACTION --> Active((ACTIVE)) Active -- READ, WRITE --> Active Active -- END TRANSACTION --> PartiallyCommitted((PARTIALLY COMMITTED)) PartiallyCommitted -- COMMIT --> Committed((COMMITTED)) PartiallyCommitted -- ABORT --> Failed((FAILED)) Committed -- ABORT --> Terminated((TERMINATED)) Failed --> Terminated </pre>	<p>05</p>
------------	--	-----------

<p>9.c</p>	<p>Need of concurrency control and recovery needed in DBMS: Problems when two simple transactions run concurrently:</p> <p>DBMS has a <i>Concurrency Control</i> sub-system to assure database remains in consistent state despite concurrent execution of transactions</p> <p>The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not, because the whole transaction is a logical unit of database processing. If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect. Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:</p> <ol style="list-style-type: none"> 1. A computer failure (system crash). A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure. 2. A transaction or system error. Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.³ Additionally, the user may interrupt the transaction during its execution. 3. Local errors or exception conditions detected by the transaction. During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception could be programmed in the transaction itself, and in such a case would not be considered as a transaction failure. 4. Concurrency control enforcement. The concurrency control may abort a transaction because it violates serializability ,or it may abort one or more transactions to resolve a state of deadlock among several transactions (see Section Transactions aborted because of serializability violations or deadlocks are typically restarted automatically at a later time. 5. Disk failure. Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction. 6. Physical problems and catastrophes. This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator. 	<p>10</p>

Two Transactions

T1

```
read_lock(Y);  
read_item(Y);  
unlock(Y);  
write_lock(X);  
read_item(X);  
X:=X+Y;  
write_item(X);  
unlock(X);
```

T2

```
read_lock(X);  
read_item(X);  
unlock(X);  
write_lock(Y);  
read_item(Y);  
Y:=X+Y;  
write_item(Y);  
unlock(Y);
```

Let's assume serial schedule S1: T1,T2

Initial values: X=20, Y=30 → Result: X=50, Y=80

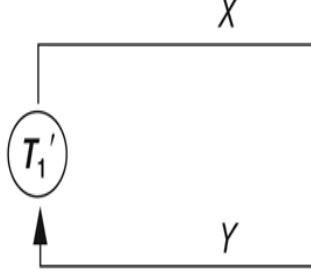
10.a

The Deadlock and starvation problem occurs when Deadlocks and Starvation

- 2PL can produce deadlocks
- Deadlock and starvation in 2PL

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.

10

	<p>(a)</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th style="width: 50px;"></th> <th style="width: 150px;">T_1'</th> <th style="width: 150px;">T_2'</th> </tr> </thead> <tbody> <tr> <td style="text-align: right; vertical-align: middle;">Time ↓</td> <td> read_lock(Y); read_item(Y); write_lock(X); </td> <td> read_lock(X); read_item(X); write_lock(Y); </td> </tr> </tbody> </table> <p style="margin-left: 40px;">(b)</p>  <p style="text-align: center;">Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).</p> <p>Deadlock prevention protocols</p> <ul style="list-style-type: none"> • Conservative 2PL, lock all needed items in advance • Ordering all items in the database <p>Possible actions if a transaction is involved in a possible deadlock situation</p> <ul style="list-style-type: none"> • Block and wait • Abort and restart • Preempt and abort another transaction <p>Two schemes that prevent deadlock (Timestamp based)</p> <p>Wait-die An older transaction is allowed to wait on a younger transaction whereas a younger transaction requesting an item from held by an older transaction is aborted and restarted with the same timestamp</p> <p>Wound-wait A younger transaction is allowed to wait on an older transaction whereas an older transaction requesting an item from held by a younger transaction preempts the younger transaction by aborting it.</p> <p>Starvation A transaction cannot proceed for an infinite period of time while other transactions in the system continue normally</p> <ul style="list-style-type: none"> • Unfair waiting scheme • Victim selection 		T_1'	T_2'	Time ↓	read_lock(Y); read_item(Y); write_lock(X);	read_lock(X); read_item(X); write_lock(Y);	
	T_1'	T_2'						
Time ↓	read_lock(Y); read_item(Y); write_lock(X);	read_lock(X); read_item(X); write_lock(Y);						
<p>10.b</p>	<p>Two phase locking techniques for concurrency control:</p> <p>Lock</p> <ul style="list-style-type: none"> • A variable associated with a data item • Describes status of the data item with respect to operations that can be performed on it <p>Types of locks</p> <ul style="list-style-type: none"> • Binary locks 	<p>10</p>						

- Locked/unlocked
 - Enforces mutual exclusion
- Multiple-mode locks:
- Each data item can be in one of three lock states
 1. Read lock or shared lock
 2. Write lock or exclusive lock
 3. Unlock

Two-Phase Locking (2PL) Protocol

Transaction is said to follow the *two-phase-locking protocol* if all locking operations precede the *first* unlock operation

- Expanding (growing) phase
- Shrinking phase

During the shrinking phase no new locks can be acquired

- Downgrading ok
- Upgrading is not

2PL Example

<p>T1'</p> <pre>read_lock(Y); read_item(Y); write_lock(X); unlock(Y); read_item(X); X:=X+Y; write_item(X); unlock(X);</pre>	<p>T2'</p> <pre>read_lock(X); read_item(X); write_lock(Y); unlock(X); read_item(Y); Y:=X+Y; write_item(Y); unlock(Y);</pre>
--	--

- **Both T1' and T2' follow the 2PL protocol**
- **Any schedule including T1' and T2' is guaranteed to be serializable**
- **Limits the amount of concurrency**
- Two-phase locking protocol (2PL)
 - All lock operations precede the first unlock operation
 - Expanding phase and shrinking phase
 - Upgrading of locks must be done in expanding phase and downgrading of locks must be done in shrinking phase
- If every transaction in a schedule follows 2PL protocol then the schedules is guaranteed to be serializable.
- Variants of 2PL
 - Basic, conservative, strict, and rigorous

--	--	--