Model Question Paper- DBMS

Q1. a)Compare DBMS and early file systems bringing out the major advantages of the database approach. (6)

Drawbacks of File system

Data redundancy: Data redundancy refers to the duplication of data. Data redundancy often leads to higher storage costs and poor access time.

Data inconsistency: Data redundancy leads to data inconsistency, lets take the same example that we have taken above, a student is enrolled for two courses and we have student address stored twice, now lets say student requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.

Data Isolation: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Dependency on application programs: Changing files would lead to change in application programs.

Data Security: Data should be secured from unauthorised access, for example a student in a college should not be able to see the payroll details of the teachers, such kind of security constraints are difficult to apply in file processing systems.

Advantages of Using the DBMS Approach

1. Controlling Redundancy:

Data redundancy (such as tends to occur in the "file processing" approach) leads to wasted storage space,duplication of effort (when multiple copies of a datum need to be updated), and a higher likelihood of theintroduction of inconsistency.

On the other hand, redundancy can be used to improve performance of queries. Indexes, for example, areentirely redundant, but help the DBMS in processing queries more quickly.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introducedwhen data is updated.

2. Restricting Unauthorized Access:

A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and tospecify account restrictions. Then, the DBMS should enforce these restrictions automatically.

3. Providing Persistent Storage for Program Objects:

Object- oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved insecondary storage so as to survive beyond program termination and to be retrievable at a later time.

4. Providing Storage Structures and Search Techniques for Efficient Query Processing:

Database systems must provide capabilities for efficiently executing queries and updates. The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

5. Providing Backup and Recovery:

A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.The recovery subsystem could ensure that the transaction is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

6. Providing Multiple User Interfaces:

Many types of users with varying levels of technical knowledge use a database, a DBMS should provide a varietyof user interfaces. For example, query languages for casual users, programming language interfaces for application programmers, forms and/or command codes for parametric users, menu- driven interfaces for standalone users.

7. Representing Complex Relationships Among Data:

A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8. Enforcing Integrity Constraints:

Most database applications have certain integrity constraints that must hold for the data.

The simplest type of integrity constraint involves specifying a data type for each data item.

A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files. This is known as a referential integrity constraint.

Another type of constraint specifies uniqueness on data item values, this is known as a key or uniqueness constraint.

9. Permitting Inferencing and Actions via Rules:

In a deductive database system, one may specify declarative rules that allow the database to infer new data!

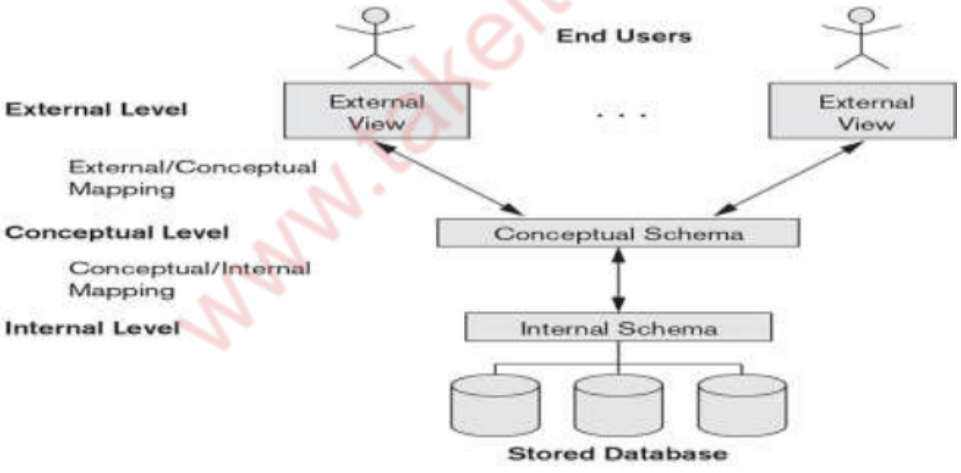
E.g., Figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

Q1 b)With a neat block diagram, explain the architecture of a typical DBMS. (10)

The Three- Schema Architecture

The goal of the three- schema architecture is to separate the user applications from the physical database



In this architecture, schemas can be defined at the following three levels:

1. **The internal level** has an internal schema,

- describes the physical storage structure of the database.

- uses a physical data model and describes the complete details of data storage

and access paths for the database.

2. **The conceptual level** has a conceptual schema,

- describes the structure of the whole database for a community of users

- hides the details of physical storage structures and concentrates on describing entities, data types,relationships, user operations, and constraints

- representational data model is used to describe the conceptual schema when a database system is

Implemented.

- This implementation conceptual schema is often based on a conceptual schema design in a high- level data model.

3. **The external or view level** includes a number of external schemas or user views,

- 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.

- As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high- level data model.

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, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings.

These mappings may be time- consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views.

Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels

Q1.c)Explain different types of user friendly interfaces and types of user who typically use each?

Types of user friendly interfaces who uses them:

Interfaces for the DBA: Interfaces for DBA include commands for creating accounts,setting system parameters,granting account authorization,changing

schema,and reorganizing the storage structure of database.

Interfaces for parametric Users: Interfaces for parametric users usually have small se of operations that they must perform repeatedly. Example: some of parametric users are bank tellers.

DBMS Interfaces:

Menu- Based Interfaces for Web Clients or Browsing. These interfaces present the user with lists of options(called menus) that lead the user through the formulation of a request.

Forms- Based Interfaces displays a form to each user. Users can fill out all of the form entries to insert new data,or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.

Graphical User Interfaces displays a schema to the user in diagrammatic form. The user then can specify a queryby manipulating the diagram. GUIs utilize both menus and forms. Most GUIs use a pointing device.

Natural Language Interfaces accepts requests written in English or some other language and attempt to understand them. Speech Input and Output use of speech as an input query and speech as an answer to a question or result. Thespeech input is detected using a library of predefined words and used to set up the parameters that are suppliedto the queries.

Interfaces for Parametric Users such as bank tellers, often have a small set of operations that they mustperform repeatedly.

Interfaces for the DBA. DBA use privileged commands. These include commands for creating accounts, settingsystem parameters, granting account authorization, changing a schema, and reorganizing the storage structuresof a database

Q2.a)Define the following with an example:

(i) Weak entity type (ii) participation constraint

(iii)cardinality ratio (iv) recursive relationship (v) specialization

Weak entity type:

- Entity types that do not have key attributes of their own are called **weak entity types**
- In contrast, regular entity types that do have a key attribute—are called

strong entity types

- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- We call this other entity type the identifying or owner entity type, and we call the relationship type thatrelates a weak entity type to its owner the identifying relationship of the weak entity type
- A weak entity type always has a total participation constraint with respect to its identifying relationshipbecause a weak entity cannot be identified without an owner entity
- A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entitiesthat are related to the same owner entity
- Assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key

Participation constraint:

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type
- This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint
- There are two types of participation constraints—total and partial
- If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance . Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called existence dependency.
- we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all
- In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship,

whereas partial participation is represented by a single line.

Cardinality ratio

Cardinality Ratios for Binary Relationships:

The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

For example, in the **WORKS_FOR** binary relationship type,**DEPARTMENT:EMPLOYEE** is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees (N),9 butan employee can be related to (work for) at most one department (1)

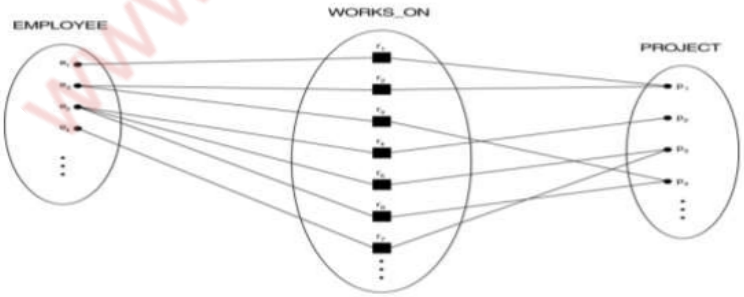
The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1,

A 1:1 relationship, MANAGES.

and M:N

In 1:1 an employee can manage at most one department and a department can have at most one manager

An M:N relationship, WORKS_ON.



In M:N an employee can work on several projects and a project can have several employees

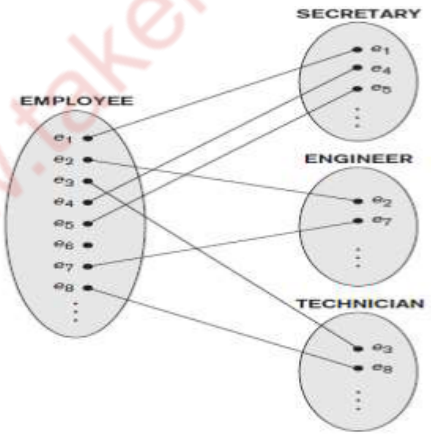
Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds

Recursive Relationships

- Each entity type that participates in a relationship type plays a particular role in the relationship
- the role name signifies the role that a participating entity from the entity type plays in each relationshipinstance For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer
- same entity type participates more than once in a relationship type in different roles, such relationship typesare called recursive relationships.

SPECIALIZATION

- Specialization is the process of defining a set of subclasses of an entity type; this entity type is calledthe superclass of the specialization.
- The set of subclasses that forms a specialization is defined on the basis of some distinguishingcharacteristic of the entities in the superclass.
- For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of thesuperclass EMPLOYEE that distinguishes among employee entities based on the job type of eachemployee.



Q2.b)Develop an ER diagram for keeping track of information about a company databasetaking into account atleast five entities.

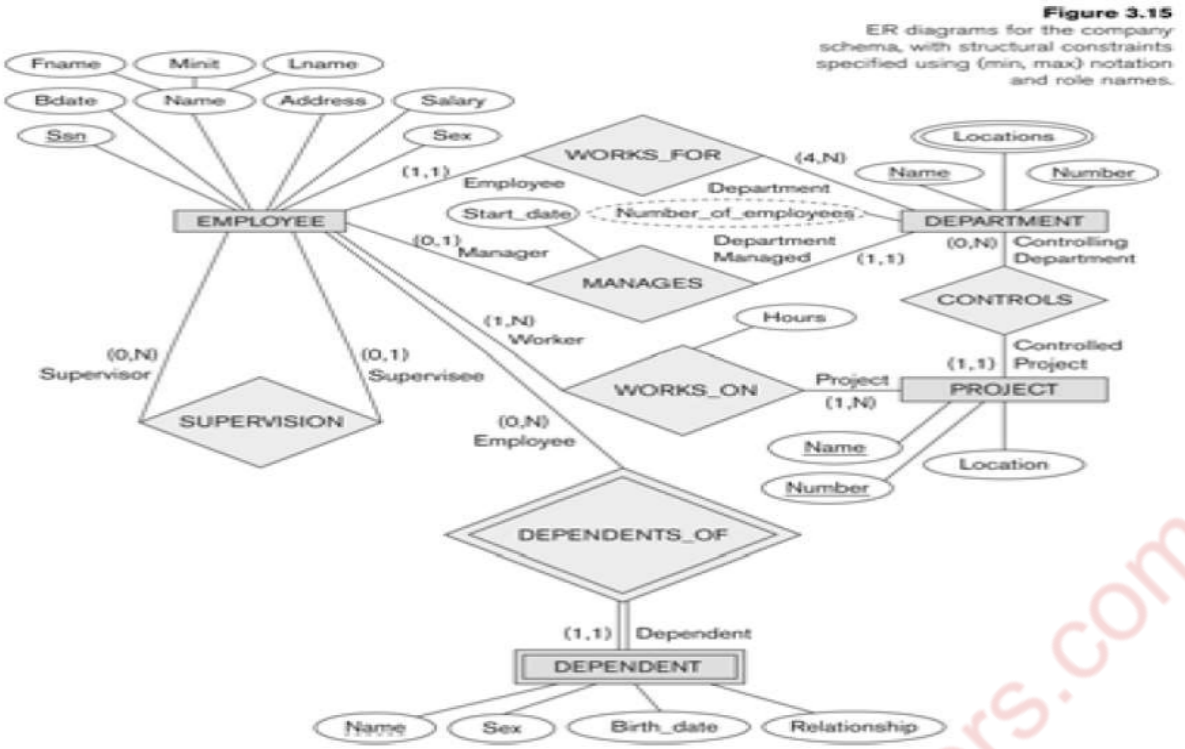


Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Q3.a)Define the following terms

i) Key ii) Super key iii) Candidate key

iv) Primary key v) Foreign key

Key:

KEYS in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Super key :

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification

Example suppose we have EmpSSN,EmpNum,Empname means EmpSSN and EmpNum name are superkeys.

Primary key:

The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a single attribute, the clause can

follow the attribute directly.

Ex: the primary key of DEPARTMENT can be specified as follows:Dnumber INT PRIMARY KEY,

Candidate key

The UNIQUE clause specifies alternate (unique) keys, also known as candidate keys

The UNIQUE clause can also be specified directly for a unique key if it is a single attribute, as in the following example: Dname VARCHAR(15) UNIQUE

FOREIGN KEY

Referential integrity is specified via the FOREIGN KEY clause,constraint can be violated whentuples are inserted or deleted, or when a foreign key or primary key attribute value is updated. The default action that SQL takes for an integrity violation is to reject the update operation thatwill cause a violation, which is known as the RESTRICT option.

The schema designer can specify an alternative action to be taken by attaching a referential triggered action clause to any foreign key constraint. The options include SET NULL, CASCADE,and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE.

In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ONDELETE and ON UPDATE: The value of the affected referencing attributes is changed to NULL for SET NULL and to the specified default value of the Key and referential integrity constraints werenot included in early versions of SQL. The action for CASCADE ON DELETE is to delete all the referencing tuples, whereas the action forCASCADE ON UPDATE is to change the value of the referencing foreign key attribute(s) to the updated (new) primary key value for all the referencing tuples.

Q3.b)Consider the following COMPANY database

EMP(Name,SSN,Salary,SuperSSN,Gender,Dno)

DEPT(DNum,Dname,MgrSSN,Dno)

DEPT_LOC(Dnum,Dlocation)

DEPENDENT(ESSN,Dep_name,Sex)

WORKS_ON(ESSN,Pno,Hours)

PROJECT(Pname,Pnumber,Plocation,Dnum)

Write the relational algebra queries for the following

(i)Retrieve the name, address, salary of employees who work for the Research department.

(ii) find the names of employees who work on all projects controlled by department number4.

iii) Retrieve the SSN of all employees who either in department no :4 or directly supervise an employee who work in department number :4

(iv) Retrieve the names of employees who have no dependents

(v) Retrieve each department number, the number of employees in the department and

their average salary.

ANSWERS:

i)Retrieve the name, address, salary of employees who work for the Research department

SELECT FNAME, LNAME, ADDRESS

FROM EMPLOYEE, DEPARTMENT

WHERE DNAME = 'Research' AND DNUMBER = DNO;

- Retrieve the name and address of all employees who work for the 'Research' department.

$$RESEARCH\_DEPT \leftarrow \sigma_{DNAME='Research'}(DEPARTMENT)$$
$$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{DNUMBER=DNO} (EMPLOYEE))$$
$$RESULT \leftarrow \pi_{FNAME,LNAME,ADDRESS}(RESEARCH\_EMPS)$$

(ii) find the names of employees who work on all projects controlled by department number4.

SELECT FNAME, LNAME

FROM EMPLOYEE

WHERE ((SELECT PNO FROM WORKS_ON WHERE SSN = ESSN)

CONTAINS

(SELECT PNUMBER FROM PROJECT

WHERE DNUM = 4));

$$DEPT5\_PROJS(PNO) \leftarrow \pi_{PNUMBER}(\sigma_{DNUM=5}(PROJECT))$$
$$EMP\_PROJ(SSN, PNO) \leftarrow \pi_{ESSN,PNO}(WORKS\_ON)$$
$$RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ \div DEPT\_PROJS$$
$$RESULT \leftarrow \pi_{LNAME,FNAME}(RESULT\_EMP\_SSNS * EMPLOYEE)$$

iii)Retrieve the SSN of all employees who either in department no :4 or directly supervise an employee who work in department number :4

(SELECT SSN FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM = DNUMBER AND MGRSSN = SSN AND DNO=4)

UNION

(SELECT SSN FROM PROJECT, WORKS_ ON, EMPLOYEE

WHERE PNUMBER = PNO AND ESSN = SSN AND DNO=4);

DEPT4(ESSN) <- - - - $\pi_{SSN(\sigma\ DN0=4(EMPLOYEE))}$

WORKER_DEPTS<- - - - - $\pi_{S\ (WORKS\text{-}\ ON\ *DEPT4)}$

MGRS<- - - - - - - - - $\pi_{DNUMBER}$(EMPLOYEE $\bowtie_{\theta SSN=MGRSSN}$ DEPARTMENT)

MANAGED_DEPTS(DNUM)<- - - - - - $\pi_{DNUMBER((\sigma\ DN0=4(MGRS))}$

RESULT<- (WORKER DEPT U MANAGED DEPS)

(iv) Retrieve the names of employees who have no dependents

SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT * FROM DEPENDENT WHERE SSN = ESSN);

$$ALL\_EMPS \leftarrow \pi_{SSN}(EMPLOYEE)$$
$$EMPS\_WITH\_DEPS(SSN) \leftarrow \pi_{ESSN}(DEPENDENT)$$
$$EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)$$
$$RESULT \leftarrow \pi_{LNAME,FNAME}(EMPS\_WITHOUT\_DEPS * EMPLOYEE)$$

V)Retrieve each department number, the number of employees in the department and their average salary.

SELECT DNO, COUNT (*), AVG (SALARY)
FROM EMPLOYEE
GROUP BY DNO;

RESULT - > Dno(COUNT(AVERAGE(SALARY)))(EMPLOYEE)

Q3 C)**Summarize the steps involved in converting the ER constructs to corresponding relational tables**

Steps in ER to Relational mapping

Step 1: Mapping of Regular Entity types

- For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes
- Include only the simple components attributes of composite attributes
- Choose one of the key attributes of E as primary key for R
- If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R

Eg: EMPLOYEE, PROJECT, DEPARTMENT are regular entities and their primary keys are Ssn, Pnumber, Dnumber respectively

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 that includes all the simple attributes of W as attributes of R
- Include as foreign key attributes of R, the primary key of relation that correspond to owner entity type
- The primary key of R is the combination of primary key of the owner and the partial key of the weak

entity type

Eg: DEPENDENT is weak entity type.

Include the primary key Ssn of EMPLOYEE which corresponds to owner entity type as foreign key attribute of DEPENDENT

Step 3: Mapping of Binary 1:1 Relationship types

- Identify the relations S and T that corresponds to the entity types participating in R

Three possible approaches:

1) **Foreign key approach**:

Choose one of the relation S and include as foreign key in S the primary key of T

Eg: MANAGES is 1:1 relationship between DEPARTMENT and EMPLOYEE

Include the primary key of EMPLOYEE as foreign key in the DEPARTMENT and rename it as Mgr_ssn

2) **Merged relationship approach**:

Merging two entity types and relationship into a single relation

3) **Cross reference approach:**

Set up third relation R for the purpose of cross referencing the primary keys of two relations S and T

Step 4: Mapping of Binary 1:N Relationship types

- Identify the relation S that represents 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 other entity type participating in R

Eg: WORKS_FOR, CONTROLS, SUPERVISION is 1:N relationship type for WORKS_FOR include primary key Dnumber of the DEPARTMENT relation as foreign key in EMPLOYEE relation and call it Dno

Step 5: Mapping of Binary M:N Relationship types

- Create a new relation S to represent R
- Include a foreign key attributes in S the primary keys of the relation that represent the participating entity types, their combination will form the primary key of S

Eg: WORKS_ON is M:N relationship type, include the primary keys of EMPLOYEE and PROJECT as foreign key for WORKS_ON relation

Step 6: Mapping of Multivalued attributes

- For each multivalued attribute A, create a new relation R
- The relation R will include attributes to A and the primary key attribute K as a foreign key in R
- DEPT_LOCATION is new relation, the attribute Dlocation represents multivalued attribute LOCATIONS of DEPARTMENT

Step 7: Mapping of Binary N- ary Relationship types

- For each n- ary relationship type R, where n>2 create a new relation S to represent R
- Include foreign key attributes in S the primary key of the relations that represent the participating entity types Also include simple attributes of n-ary relationship types

Q4 a)Explain with example basic constraints that can be specified when a database table is created.

Constraints can be divided into the following two types, Column level constraints: Limits only column data.

Table level constraints: Limits whole table data. Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

CREATE TABLE Student(s_id int NOT NULL, Name varchar(60), Age int);

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value.

CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);

Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables.

```
CREATE table Order_Detail(
   order_id int PRIMARY KEY,
   order_name varchar(60) NOT NULL,
   c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id)
);
```

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database

CREATE table Student(s_id int NOT NULL CHECK(s_id > 0), Name varchar(60) NOT NULL,Age int);

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

Q4 b)Write SQL syntax for the following with example:

(i) SELECT
(ii) ALTER
(iii) UPDATE
I) SQL SELECT Statement Syntax
SELECT column1, column2....columnN

FROM table_nameWHERE CONDITION;

Example: SELECT * FROM Customers

WHERE Country='Mexico';

UPDATE Syntax:

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

Example:UPDATE CUSTOMERS SET ADDRESS = 'Pune'

WHERE ID = 6;

ALTER TABLE SYNTAX:

ALTER TABLE table_name

ADD column_name datatype;

Example:ALTER TABLE Customers

ADD Email varchar(255);

Q4 c)Consider the following relation schema

Works(Pname,Cname,salary)

Lives(Pname,Street,City)

Located_in (Cname, city)

Manager(Pname,Mgrname)

Write the SQL queries for the following

i) Find the names of all persons who live in the city Bangalore.

ii) Retrieve the names of all person of "Infosys" whose salary is between Rs .50000

iii)Find the names of all persons who lives and work in the same city

iv)List the names of the people who work for “Tech M” along with the cities they live in.

v)Find the average salary of “Infosys” persons

i)SELECT L. Pname,City from Works W and Lives L

whereW.pname=L.pname andL.city=Banglore;

ii)SELECT L.Pname from Works W,Lives L where W.cnmae=’infosys’ AND

W.Pname=L.Cname and salary <50,000

iii)SELECT W.Pname FROM Works W,LivesL,LocatedIN I FROM Works w,LivesL,Locatedin I WHERE

W.Cname=I.Cname AND W.Pname=L.Pname AND I.City=L.City;

iv) SELECT L.PNAME,City FROM Works W,LivesLwhereCname="Tech M" AND W.Pname=L.Pname;

V)SELECT AVG(Salary) From Works W L.Located_in Where W.Cname=I.Cname AND L.Cname=Infosys;

Q 5a.Explain the following constructs used in SQL with example:

(i)Nested queries

ii)Aggregate functions

iii) Triggers

iv)Views and their updability

v) schema change statements

TRIGGERS:

- Trigger has three components: event, condition, action

Event: When event happens, trigger is activated

Condition: If condition is true, trigger executes or skips

Action: Actions are performed by triggers

- When event occurs and condition is true, execute the action

Syntax:

CREATE TRIGGER <trigger_name>

[Before|After|Instead of]

[Insert|Update|Delete] on <table_name>

[for each row]

[when<condition to execute>]

Trigger body;

- Rather than offering users only the option of aborting an operation (using ASSERTION) that causes a violation, the DBMS should make the following option available.
- It may be useful to specify a condition that, if violated, causes some user to be informed of the violation.

  eg: A manager may want to be informed if an employee's travel expenses exceed a certain limit by receiving a message whenever this occurs. The action that the DBMS must take in this case is to send an appropriate message to that user. The condition is thus used to monitor the database.

View

A view is a single table that is derived from other tables.

- These other tables could be base tables or previously defined views.
- A view does not necessarily exist in physical form; it is considered a virtual table, in contrast to base tables, whose tuples are actually stored in the database.

Syntax:

CREATE VIEW <view name> [(<column name>{,<column>})]

AS <select statement>

- For example, we may frequently issue the following query that retrieve the employee name and the project names that the employee works on.

SELECT FNAME, PNAME

FROM EMPLOYEE, PROJECT, WORKS_ON WHERE (SSN = ESSN AND PNO = PNUMBER);

- Rather than having to specify the join of the EMPLOYEE, WORKS_ON, and PROJECT tables every time weissue that query, we can define a view that is a result of these joins.
- We can then issue queries on the view, which are specified as single- table retrievals rather than asretrievals involving two joins on three tables.
- We callthe EMPLOYEE, WORKS_ON, and PROJECT tables the defining tables of the view.

```
V1: CREATE VIEW   WORKS_ON1
    AS  SELECT    FNAME, LNAME, PNAME, HOURS
        FROM      EMPLOYEE, PROJECT, WORKS_ON
        WHERE     SSN=ESSN AND PNO=PNUMBER;

V2: CREATE VIEW   DEPT_INFO(DEPT_NAME,NO_OF_EMPS,TOTAL_SAL)
    AS  SELECT    DNAME, COUNT (*), SUM (SALARY)
        FROM      DEPARTMENT, EMPLOYEE
        WHERE     DNUMBER=DNO
        GROUP BY  DNAME;
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

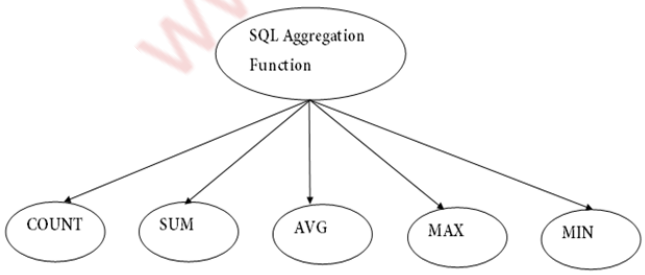
DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL

- We did not specify any new attribute names for the view WORKS_ON1; in this case, WORKS_ON1 inherits the names of the view attributes from the defining tables EMPLOYEE, PROJECT, and WORKS_ON.
- We explicitly specifies new attribute names for the view DEPT_INFO, using a one- to- one correspondence between the attributes specified in the CREATE VIEW clause and those specified in the SELECT clause ofthe query that defines the view.
- We can now specify SQL queries on views V1 and V2 in the same way we specify queries involving base tables.

Aggregate Functions

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value. It is also used to summarize the data.



COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non- numeric data types.

SELECT COUNT(column_name) FROM table_name WHERE condition;

SELECT COUNT(ProductID) FROM Products;

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

SELECT AVG(Price) FROM Products;

returns the total sum of a numeric column.

The SUM() function

SELECT sum(Price) FROM Products;

the **SQL MIN() and MAX() Functions**

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

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

SELECT MIN(Price) AS SmallestPrice FROM Products;

Schema change statements

The DROP Command:

- The DROP command can be used to drop named schema elements, such as tables, domains, or constraints.
  - There are two drop behavior options: CASCADE and RESTRICT.
  - For example, to remove the COMPANY database schema and all its tables, domains, and other elements,the CASCADE option is used as follows:

DROP SCHEMA COMPANY CASCADE;

  - If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has no elements init; otherwise, the DROP command will not be executed
  - If a base table within a schema is not needed any longer, the relation and its definition can be deleted byusing the DROP TABLE command.
  - For example, if we no longer wish to keep track of dependents of employees in the COMPANY database :

DROP TABLE DEPENDENT CASCADE;

  - If the RESTRICT option is chosen instead of CASCADE, a table is dropped only if it is not referenced in anyconstraints (for example, by foreign key definitions in another relation) or views.
  - With the CASCADE option, all such constraints and views that reference the

table are droppedautomatically from the schema, along with the table itself.

The ALTER Command:

- The definition of a base table or of other named schema elements can be changed by using the ALTERcommand.
- For base tables, the possible alter table actions include :

Adding or dropping a column (attribute)

Changing a column definition

Adding or dropping table constraints.

- For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relationsin the COMPANY schema, we can use the command:

ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN job VARCHAR(12);

- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
- If CASCADE is chosen, all constraints and views that reference the column are dropped automatically fromthe schema, along with the column.
- If RESTRICT is chosen, the command is successful only if no views or constraints (or other elements)reference the column.
- The following command removes the attribute ADDRESS from the EMPLOYEE base table:

ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

Nested queries

In nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like **IN, NOT IN, ANY, ALL** etc are used in writing independent nested queries.

Example: SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID

FROM CUSTOMERS WHERE SALARY > 4500) ;

Example: INSERT INTO CUSTOMERS(SELECT * FROM CUSTOMERS

WHERE ID IN (SELECT ID FROM CUSTOMERS) ;

Example: DELETE FROM CUSTOMERS

WHERE AGE IN (SELECT AGE FROM CUSTOMERS WHERE AGE >= 27);

Q5 b)What is Dynamic SQL? How it is different from Embedded SQL

Dynamic SQL

- Programming technique that enables to build SQL statements dynamically at runtime
- This can be helpful when it is necessary to write code that can adjust to varying databases, conditions, orservers. It also makes it easier to automate tasks that are repeated many times
- Two main commands: PREPARE and EXECUTE
- Eg: char c_sqlstring[] = {"DELETE FROM Sailors WHERE rating>5"};

EXEC SQL PREPARE readytogo FROM :csqlstring;

EXEC SQL EXECUTE readytogo;

- The first statement declares the C variable c_sqlstring and initializes its value to the string representation of an SQL command
- The second statement results in this string being parsed and compiled as an SQL command, with the resultingexecutable bound to the SQL variable readytogo.
- The third statement executes the command

Static (Embedded) SQL	Dynamic (Interactive) SQL
In Static SQL, how database will be accessed is predetermined in the embedded SQL statement.	In Dynamic SQL, how database will be accessed is determined at run time.
It is more swift and efficient.	It is less swift and efficient.
SQL statements are compiled at compile time.	SQL statements are compiled at run time.
Parsing, Validation, Optimization and	Parsing, Validation, Optimization and

Generation of application plan are done at compile time.	Generation of application plan are done at run time.
It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non uniformly.
EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.
It is less flexible.	It is more flexible.

Limitation of Dynamic SQL:

We cannot use some of the SQL statements Dynamically.

Performance of these statements is poor as compared to Static SQL.

Limitations of Static SQL:

They do not change at runtime thus are hard- coded into applications.

Q5c) Consider the following COMPANY database (10 marks)

EMP(Name,SSN,Salary,SuperSSN,Dno)

DEPT(DNum,Dname,MgrSSN,Dno)

DEPT_LOC(Dnum,Dlocation)

DEPENDENT(ESSN,Dep_name,Sex)

WORKS_ON(ESSN,Pno,Hours)

PROJECT(Pname,Pnumber,Plocation,Dnum)

Write the SQL queries for the following

i) Retrieve the name of the employee who works with same department as ravi

ii)Retrieve the number of dependents for an employee “Ravi”

iii)Retrieve the name of the managers working in location “DELHI”who has no female dependents

iv)List female employees from Dno=20 earning more than 50000

v)List “CSE” department details

1)SELECT name,dno FROM employee e,department d where dno IN (SELECT dno FROM emp WHERE name='ravi';

ii)SELECT COUNT(*) FROM dependents d,emp e WHERE e.ssn=d.essn AND name='ravi';

iii)SELECT E.NAME FROM EMPLOYEE E ,DEPARTMENT D,DEPTLOCATIONS DL WHERE E.SSN=D.MGRSSN AND D.DNO=DL.DNO AND DLOCATION='DELHI' where NOT EXISTS(SELECT FROM employee where d.dnumber=e.dno and d.sex='F'

Iv) select ssn from emp where dno=20 and salary<500000

v) SELECT * FROM dept where dname='cse';

Q6 a). What is SQLJ? How it is different from JDBC.

SQLJ

- SQLJ(short for SQL- Java) developed by group of database vendors and Sun
- It was developed to complement the dynamic way of creating queries in JDBC with a static model
- It is very close to embedded SQL
- It has semi- static SQL queries allows the compiler to perform SQl syntax checks, strong type checks ofthe compatibility of the host variables with respective SQL attributes, views and stored procedure atcompliation time.

	SQLJ (static)	JDBC (dynamic)
PERFORMANCE	static SQL is faster than dynamic SQL,	Dynamic SQL statements require the SQL statements to be parsed, table/view authorization to be checked,
AUTHORIZATION	With SQLJ, the owner of the application grants EXECUTE authority on the plan or package, and the recipient of that GRANT must run the application	With JDBC, the owner of the application grants privileges on all the underlying tables that are used by the application.

Q6 b) Draw and explain 3- tier Architecture and technology relevant to each tier.

Write the advantages of 3- tier architecture.

The three tier application architecture

Data intensive internet applications can be understood in terms of three functional components

Data management, application logic and presentation

Single Tier and Client- Server Architectures

Data intensive internet applications were combined into a single tier, including DBMS, application logic and the user interface.

The application typically ran on a mainframe and users accessed it through dumb terminals that could peform only data input and display

Advantage: easily maintained by a central administrator

Disadvatages:

- users expect GUI that require more computational power than simple dumb terminals

- do not scale for thousands of users

The commoditization of PC and the availability of cheap clients computers led to the development of the two tier architecture.

Two tier architecture also referred as client server architecture, consists of client computer and servercomputer, which interacts with through a well defined protocols.

Clients implement just the graphical user interface and the server implements both the business logic and the data management, such clients are called as Thin Clients.

Clients that implement both user interface and business logic, with the remaining part being implemented at the server level, such clients are called as Thick Clients.

Thick clients model has disadvantages over Thin clients model No central place to update and maintain the business logic, sinc the application runs at many sites - Large amount of trust is required between client and server.eg: ATM - Does not scale with the number of clients - Do not scale as the application accesses more and more database systems

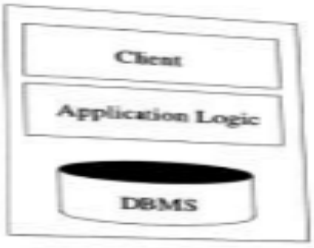


Figure 7.5 A Single-Tier Architecture

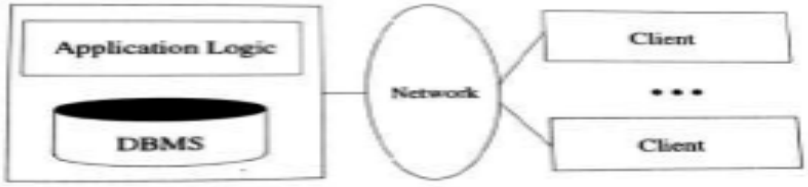


Figure 7.6 A Two-Server Architecture: Thin Clients

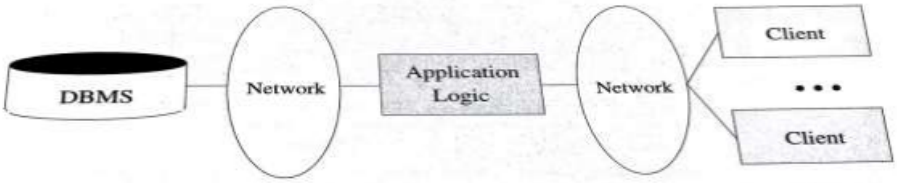


Figure 7.8 A Standard Three-Tier Architecture

Three Tier Architecture

three tier architecture goes one step further seperates application logic from data management

- ➤ Presentation Tier:

- user requires a natural interface to make requests provide input and to see results

- this layer of code is easy to adopt to different display devices and formats

- it decides how to display the information

- eg: web based interfaces

- ➤ Middle Tier:

- Application logic executes here

- It controls what data needs to be input before an action can be executed, determines the

control flow between multi action steps, controls access to DB layer, assembles dynamically generated HTML pages from DB query

- ➤ Data Management Tier:

- Data intensive web applications involve DBMS

- Information is stored and retrieved from DB

- Information is passed back to the logic tier for processing and eventually back to the user

- Technologies used are XML, stored procedures

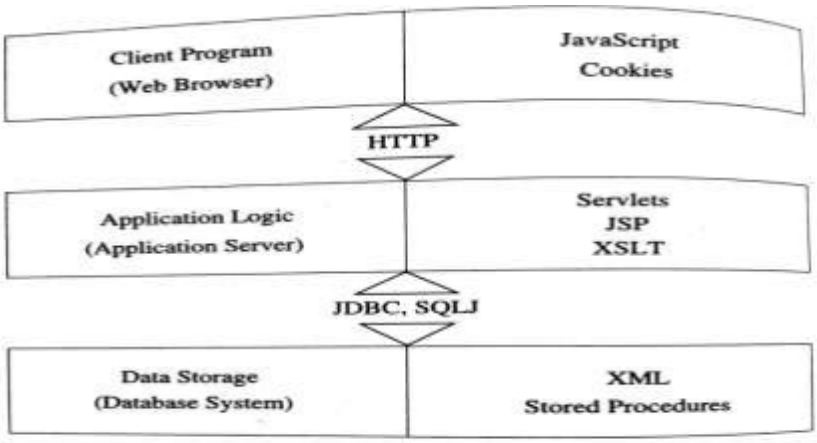


Figure 7.9 Technologies for the Three Tiers

Advantages of Three Tier Architecture

Heterogeneous Systems: Applications can utilize the strengths of different platforms and differentsoftware components at the different tiers. It is easy to modify or replace the code at any tier without affecting the other tiers.

Integrated data access: In many applications, the data must be accessed from several sources. This can be handled transparently at the middle tier, where it can centrally manage the **connections to all** DB systems involved.

Scalability to Many Clients:

Each client is lightweight and all access to the system is through the middle tier. The middle tier can share database connections across clients, and if the middle tier becomes the bottle- neck, we can deploy several servers executing the middle tier code; clients can connect to anyone of these servers, if the logic is designed appropriately.

Software Development Benefits: By dividing the application cleanly into parts that address presentation, data access, and business logic, we gain many advantages. The business logic is centralized, and is therefore easy to maintain, debug, and change. Interaction between tiers occurs through well- defined, standardized APIs. Therefore, each application tier can be built out of reusable components that can be individually developed, debugged, and tested.

Q6 Explain stored procedure with example

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

Stored procedures are useful in the following circumstances:

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

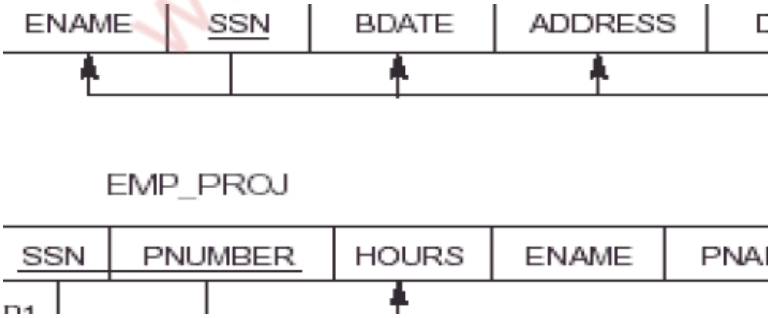
b) Executing a program at the server can reduce data transfer and hence communication cost between the client and server in certain situations.

c) These procedures can enhance the modeling power provided by views by allowing more complex types of derived data to be made available to the database users. SELECT C.cid, C.cname, COUNT(*) FROM Customers C, Orders a WHERE C.cid = O.cid GROUP BY C.cid, C.cname

Q7 a)Explain insertion, deletion and modification anomalies. Why are they considered bad? Illustrate with example

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. Reducing redundant values in tuples. Save storage space and avoid update anomalies.

- Insertion anomalies.
- Deletion anomalies.
- Modification anomalies



Insertion Anomalies

To insert a new employee tuple into EMP_DEPT, we must include either the

attribute values for that department that the employee works for, or nulls. It's difficult to insert a new department that has no employee as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity - not a department entity.

Deletion Anomalies

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 lost from the database.

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.

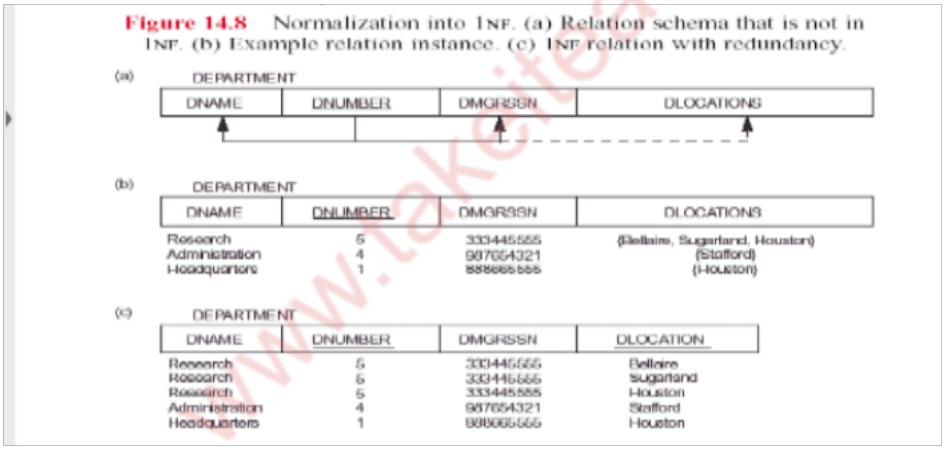
Q7 b) Explain 1NF, 2NF, 3NF with example.

First Normal Form (1NF)

It states that the domains of attributes must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. It disallows multivalued and composite attribute.

Practical Rule: "Eliminate Repeating Groups," i.e., make a separate table for each set of related attributes, and give each table a primary key.

Formal Definition: A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only.

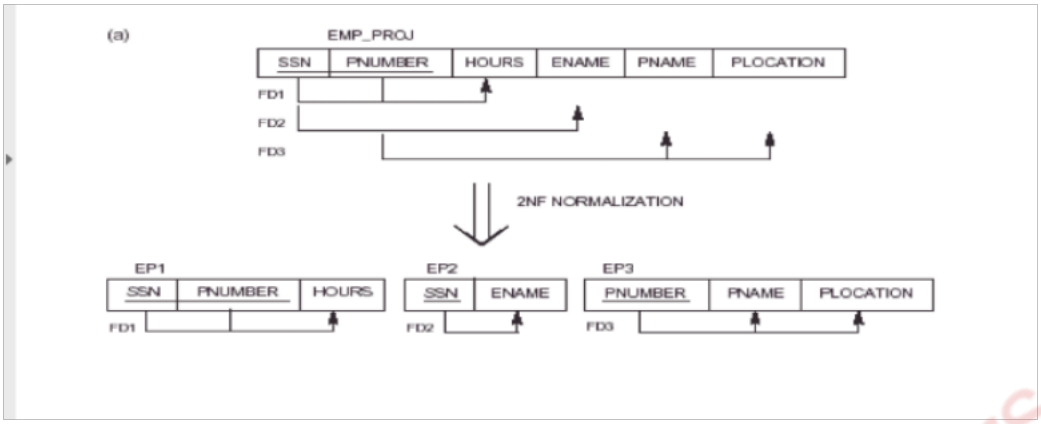


Second Normal Form (2NF)

Second normal form is based on the concept of fully functional dependency. A functional X- >Y is a fully functional dependency is removal of any attribute A from X means that the dependency does not hold any more. A relation schema is in 2NF if every nonprime attribute in relation is fully functionally dependent on the primary key of the relation. It also can be restated as: a relation schema is in 2NF if every

nonprime attribute in relation is not partially dependent on any key of the relation.Practical Rule: "Eliminate Redundant Data," i.e., if an attribute depends on only part of a multivalued key, remove it to a separate table.

Formal Definition: A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key

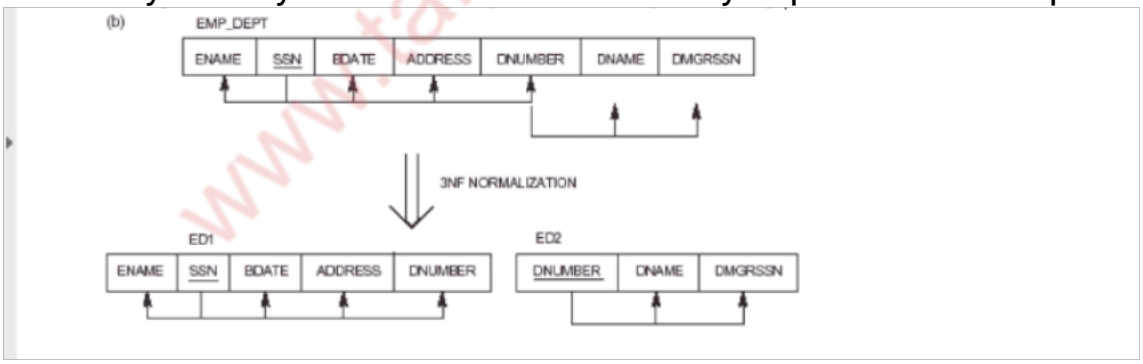


Third Normal Form (3NF)

Third normal form is based on the concept of transitive dependency. A functional dependency X Y in a relation is a transitive dependency if there is a set of attributes Z that is not a subset of any key of the relation, and both X Z and Z Y hold. In other words, a relation is in 3NF if, whenever a functional dependency X A holds in the relation, either (a) X is a superkey of the relation, or (b) A is a prime attribute of the relation.

Practical Rule: "Eliminate Columns not Dependent on Key," i.e., if attributes do not contribute to a description of a key, remove them to a separate table.

Formal Definition: A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.



1NF: R is in 1NF iff all domain values are atomic.
2NF: R is in 2 NF iff R is in 1NF and every nonkey attribute is fully dependent on the key.
3NF: R is in 3NF iff R is 2NF and every nonkey attribute is non- transitively dependent on the key.

7c) Consider the relation schema R(A,B,C,D,E,F) and the functional dependencies

A- >B,C- >DF,AC- >E, D- >F.What is the primary key of this relation R?What is its highest normal form?Preserving the dependency,decompose R into third normal form.

Steps to find the highest normal form of a relation:

1. Find all possible candidate keys of the relation.
2. Divide all attributes into two categories: prime attributes and non- prime attributes.
3. Check for 1st normal form then 2nd and so on. If it fails to satisfy nth normal form condition, highest normal form will be n- 1.

Step 1. As we can see, $(AC)^+$ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C or D can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2. Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non- prime {B,D,E} in this example.

The relation is not in 2nd normal form because A- →B(A is proper subset of candidate key AC) and C- >DF is not in 2nd normal form (C is proper subset of candidate key AC) and D→F is in 2nf because(D is not a proper subset of candidate key AC).

The relation is not in 2nd Normal form because A- >B is partial dependency (A which is subset of candidate key AC is determining non- prime attribute B) and 2nd normal form does not allow partial dependency.

The relation is not in 3rd normal form because in A→B(neither A is a super key nor B is a prime attribute) and in C→DF (neither C is a super key nor DF is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form will be 1st Normal Form

8a) Define non- additive join property of a decomposition and write an algorithm for testing of non additive join property

Dependency Preservation Property enables us to enforce a constraint on the original relation from corresponding instances in the smaller relations. If we decompose a relation R into R1 &R2 all dependencies of R must be apart of R1 & R2 or must be derivable from combination of FD's of R1 & R2

Lossless join property enables us to find any instance of the original relation from corresponding instances in the smaller relations (Both used by the design algorithms to achieve desirable decompositions). A property of decomposition, which ensures that no spurious rows are generated when relations are reunited through a natural join operation.

Algorithm 11.1 Testing for the lossless (nonadditive) join property
Input: A universal relation R, a decomposition DECOMP = {R1 ; R2; : : : ; Rn } of R, and a set F of functional dependencies
. 1. Create an initial matrix S with one row i for each relation Ri in DECOMP , and one column j for each attribute Ajin R.
2. Set S(i; j) := bijfor all matrix entries. 3. For each row i for each column j if Ri includes attribute AjThen set S(i; j) := aj
4. Repeat the following loop until a complete loop execution results in no changes to S For each X - > Y in
F for all rows in S which has the same symbols in the columns corresponding to attributes in X make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: if any of the rows has an "α" symbol for the column, set the other rows to the same "α" symbol in the column. If no "α" symbol exists for the attribute in any of the rows, choose one of the β symbols that appear in one of the rows for the attribute and set the other rows to that same symbol β in the column
5. If a row is made up entirely of "α" symbols, then the decomposition has the lossless join property; otherwise it does not.

8c) Write the algorithm to find the minimal cover for a sets of FD's

Consider R={A,B,C,D,E,F} .FD's {A- >C,AC- >D,E- >AD,E- >H} Find the irreducible cover for this set of FD's(minimal cover)
Solution:

1. **Break down** the RHS of each **functional dependency** into a **single attribute**

F={A→C
AC→D
E→A
E- >D
E→H

Next: minimize the LHS

Trying to remove *unecessary* attributes from the LHS of each **functional dependency**

NOTE: you *only* need to consider **functional dependencies** whose LHS has 2 or more attributes !

AC → D: Check if A is neccessary
Replace "AC → D" with "C → D" and compute C^+
If $D \in C^+$, then A is unnecessary

Compute C^+:

$C^+ = C$ - No

C → D: Check if C is neccessary
Replace "AC → D" with "A → D" and compute A^+
If $D \in A^+$, then C is unnecessary

Compute A^+:

$A^+ = A$ (A → C)
$= AC$ (AC → D)
$= ACD$ - Done - YES !

Result:

°F = { A → C
A → D
E → A
E → D
E → H }

Next: minimize the *set* of functional dependencies

Trying to remove *unecessary* functional dependencies

°F = { A → C
A → D
E → A
E → D
E → H }

---

1. "A → C": is unnecessary if A^+ contains C without using A → C

$A^+ = A$ (A → D)
$= AD$ Done

A → C is necessary

---

2. "A → D": is unnecessary if A^+ contains D without using A → D

$A^+ = A$ (A → C)
$= AC$ Done

A → D is necessary

---

3. "E → A": is unnecessary if E^+ contains A without using E → A

E^+ = E (E → D)
= DE (E → H)
= DEH Done

E → A is necessary

---

3. "E → D": is unnecessary if E^+ contains D without using E → D

E^+ = E (E → A)
= AE (E → H)
= AEH (A → C)
= ACEH (A → D)
= ACDEH Done

E → D is unnecessary !!!

---

3. "E → H": is unnecessary if E^+ contains H without using E → H

E^+ = E (E → A)
= AE (E → D)
= ADE (A → C)
= ACDE (A → D)
= ACDE Done

E → D is necessary

---

1. **Result:**

F= { A → C
A → D
E → A
E → H }

2. Group the **functional dependencies** that have common LHS together

Final Result:

$$F = \{ A \rightarrow CD,\ E \rightarrow AH \}$$

The resulting set of **functional dependencies** is a **minimal cover** of the ***original set*** **of functional dependencies**

The **minimal cover** is **equivalent** to the ***original set*** **of functional dependencies** but **may have** a fewer number of **functional dependencies**

8c) Given below are two sets of FD's for a relation R(A,B,C,D,E).Are they equivalent? F={A- >C,AC- >D,E- >AD,E- >H} and G={A- >CD,E- >AH}

F COVERS G	G COVERS F
A+= {A CD}	A+ ={ A C D}
E+= { E A H CD}	AC+={AC D}
	E+={ E AC D H}

Determining whether F covers G-

- $(A)^+ = \{ A , C , D \}$ // closure of left side of A → CD using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of E → AH using set F

Thus, we conclude F covers G i.e. F ⊇ G

Determining whether G covers F-

- $(A)^+ = \{ A , C , D \}$ // closure of left side of A → C using set G
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of AC → D using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of E → AD and E → H using set G

Determining whether both F and G cover each other-

From Step- 01, we conclude F covers G.

From Step- 02, we conclude G covers F.

Thus, we conclude both F and G cover each other i.e. F = G.

Q9 a)Briefly discuss on the two phase locking protocol used in concurrency control. How does it

it gurantees serializability.

Two- Phase Locking Techniques for Concurrency Control

- Some of the main techniques used to control concurrent execution of transactions are based on theconcept of locking data items.
- A **lock** is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.

Types of Locks and System Lock Tables

Binary Locks.

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

requests the item.

- If the value of the lock on *X* is 0, the item can be accessed when requested, and the lock value

is changed to 1. We refer to the current value (or state) of the lock associated with item *X* as

lock(*X*).

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

  If LOCK(*X*) = 1, the transaction is forced to wait.

  If LOCK(*X*) = 0, it is set to 1 (the transaction **locks** the item) and the transaction is allowed to access item *X*.

- When the transaction is through using the item, it issues an **unlock_item(*X*)** operation, which sets LOCK(*X*) back to 0 (**unlocks** the item) so that *X* may be accessed by other transactions. Hence, a binary lock enforces **mutual exclusion** on the data item.
- A description of the lock_item(*X*) and unlock_item(*X*) operations is shown in Figure.
- The system needs to maintain *only these records for the items that are currently locked* in a **lock table**, which could be organized as a hash file on the item name. Items not in the lock table are considered to be unlocked. The DBMS has a **lock manager subsystem** to keep track of and control access to locks.

The simple binary locking scheme described here is used; every transaction must obey the following rules:

1. A transaction *T* must issue the operation lock_item(*X*) before any read_item(*X*) or write_item(*X*)

operations are performed in *T.*

2. A transaction *T* must issue the operation unlock_item(*X*) after all read_item(*X*) and write_item(*X*)

operations are completed in *T.*

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

4. A transaction *T* will not issue an unlock_item(*X*) operation unless it already

holds the lock on item *X*.
These rules can be enforced by the lock manager module of the DBMS. Between the lock_item(*X*)
and unlock_item(*X*) operations in transaction *T*, *T* is said to **hold the lock** on item *X*. At most one
transaction can hold the lock on a particular item. Thus no two transactions can access the same item
concurrently.

```
lock_item(X):
B:  if LOCK(X) = 0                  ("item is unlocked")
        then LOCK(X) ←1             ("lock the item")
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                    (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Shared/Exclusive (or Read/Write) Locks.

- The preceding binary locking scheme is too restrictive for database items because at most

one transaction can hold a lock on a given item. We should allow several transactions to
access the same item *X* if they all access *X* for *reading purposes only*.

- This is because read operations on the same item by different transactions are *not conflicting*.

However, if a transaction is to write an item *X*, it must have exclusive access to *X*.

- For this purpose, a different type of lock, called a **multiple- mode lock**, is used. In this

scheme—called **shared/exclusive** or **read/write** locks—there are three locking operations:
read_lock(*X*), write_lock(*X*), and unlock(*X*).

- A lock associated with an item *X*, LOCK(*X*), now has three possible states: *read- locked*, *writelocked*, or *unlocked*. A **read- locked item** is also called **share- locked** because other transactions are allowed to read the item, whereas a **write- locked item** is called **exclusive- locked** because a single transaction exclusively holds the lock on the item.
- The three operations read_lock(*X*), write_lock(*X*), and unlock(*X*) are described in Figure As before, each of the three locking operations should be considered indivisible; no interleaving shouldbe allowed once one of the operations is started until either the operation terminates by granting the lock or the transaction is placed in a waiting queue for the item.

The shared/exclusive locking scheme, the system must enforce the following

rules:
1. A transaction *T* must issue the operation read_lock(*X*) or write_lock(*X*) before any read_item(*X*)
operation is performed in *T.*
2. A transaction *T* must issue the operation write_lock(*X*) before any write_item(*X*) operation is
performed in *T.*
3. A transaction *T* must issue the operation unlock(*X*) after all read_item(*X*) and write_item(*X*)
operations are completed in *T.*
4. A transaction *T* will not issue a read_lock(*X*) operation if it already holds a read (shared) lock or a
write (exclusive) lock on item *X*. This rule may be relaxed for downgrading of locks, as we discuss
shortly.
5. A transaction *T* will not issue a write_lock(*X*) operation if it already holds a read (shared) lock or write(exclusive) lock on item *X*. This rule may also be relaxed for upgrading of locks, as we discuss shortly.
6. A transaction *T* will not issue an unlock(*X*) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item *X*.

- A transaction is said to follow the **two- phase locking protocol** if *all* locking operations (read_lock, write_lock) precede the *first* unlock operation in the transaction.
- Such a transaction can be divided into two phases: an **expanding** or **growing (first) phase**, during

which new locks on items can be acquired but none can be released; and a **shrinking (second) phase**,
during which existing locks can be released but no new locks can be acquired.

- If lock conversion is allowed, then upgrading of locks (from read- locked to write- locked) must be done during the expanding phase, and downgrading of locks (from write- locked to read- locked) must be done in the shrinking phase.
- Transactions *T*1 and *T*2 in Figure 5.11(a) do not follow the two- phase locking protocol because the write_lock(*X*) operation follows the unlock(*Y*) operation in *T*1, and similarly the write_lock(*Y*) operation follows the unlock(*X*) operation in *T*2.
- If we enforce two- phase locking, the transactions can be rewritten as *T*1' and *T*2', as shown in Figure5.12. Now, the schedule shown in Figure 5.11(c) is not permitted for *T*1' and *T*2' under the rules of locking because *T*1' will issue its write_lock(*X*) *before* it unlocks item *Y*; consequently, when *T*2' issues its read_lock(*X*), it is forced to wait until *T*1' releases the lock by issuing an unlock (*X*) in the schedule

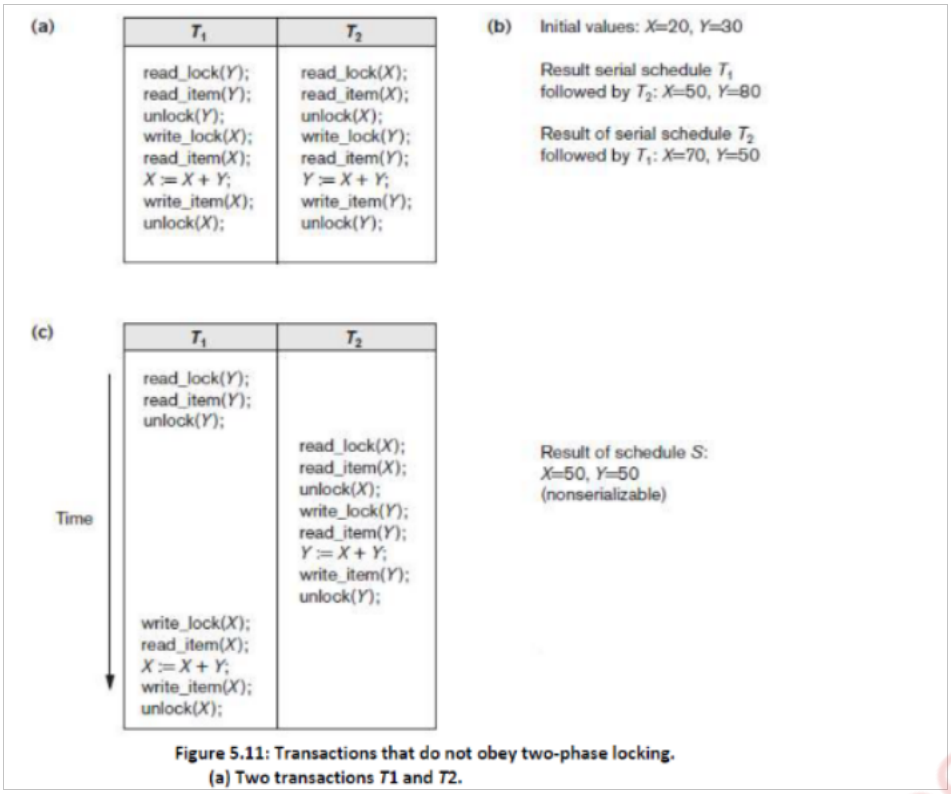


Figure 5.11: Transactions that do not obey two-phase locking.
(a) Two transactions T1 and T2.

Basic, Conservative, Strict, and Rigorous Two- Phase Locking.

There are a number of variations of two- phase locking (2PL). The technique just described is known as **basic 2PL**.

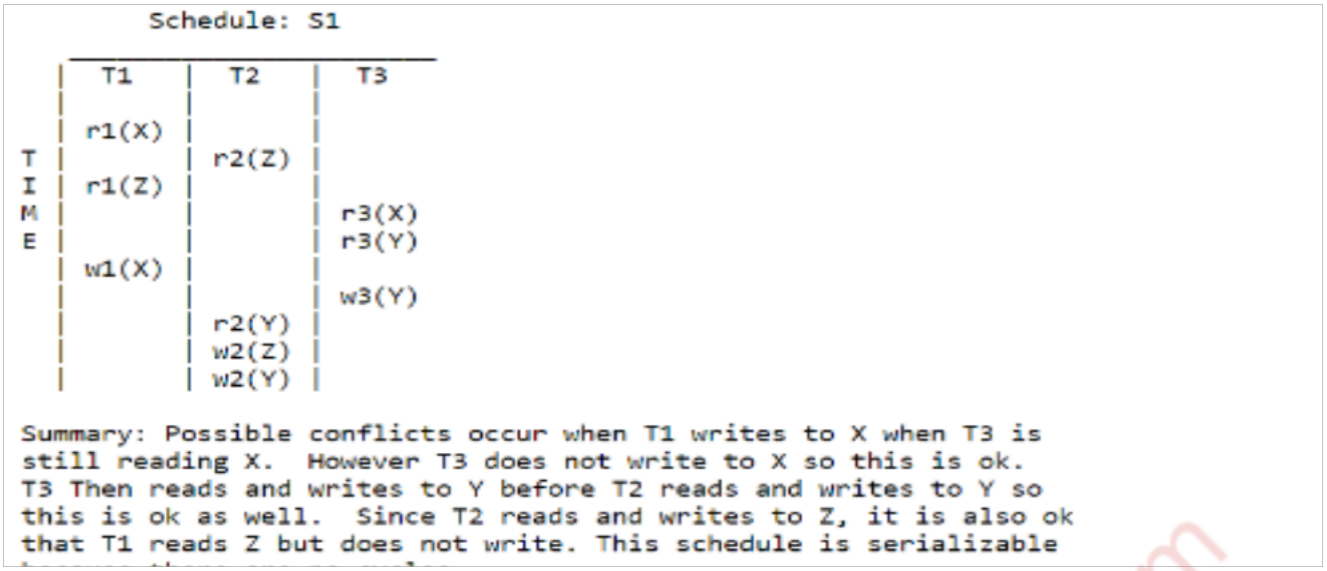
- A variation known as **conservative 2PL** (or **static 2PL**) requires a transaction to lock all the items it accesses *before the transaction begins execution*, by **predeclaring** its *read- set* and *write- set*.
- The **read- set** of a transaction is the set of all items that the transaction reads, and the **write- set** is the set of all items that it writes. If any of the predeclared items needed cannot be locked, the transaction does not lock any item; instead, it waits until all the items are available for locking.
- The difference between strict and rigorous 2PL: the former holds write- locks until it commits,

whereas the latter holds all locks (read and write).

- Also, the difference between conservative and rigorous 2PL is that the former must lock all its items *before it starts*, so once the transaction starts it is in its shrinking phase; the latter does not unlock any of its items until *after it terminates* (by committing or aborting), so the transaction is in its expanding phase until it ends.
- Usually the **concurrency control subsystem** itself is responsible for generating the read_lock and write_lock requests.

9 b)Check whether given schedule is serializable or not using precedence graph. Explain with algorithm.

S1:R1(X) R2(Z) R1(Z) R3(X) R3(Y) W1(X) W3(Y) R2(Y) W2(Z) W2(Y)



9 c)Explain Basic time stamping algorithm Basic Timestamp Ordering (TO).

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

If T is aborted and rolled back, any transaction T1 that may have used a value written by T must also be rolled back. Similarly, any transaction T2 that may have used a value written by T1 must also be rolled back, and so on. This effect is known as cascading rollback and is one of the problems associated with basic TO, since the schedules produced are not guaranteed to be recoverable.

The concurrency control algorithm must check whether conflicting operations violate the timestamp
ordering in the following two cases:

1. Whenever a transaction T issues a write_item(X) operation, the following check is performed:
   a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.
   b. If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).
2. Whenever a transaction T issues a read_item(X) operation, the following check is performed:
   a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done be cause some younger transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the

value of item X before T had a chance to read X.

b. If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X)

Strict Timestamp Ordering (TO). A variation of basic TO called strict TO ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable. In this variation, a transaction T issues a read_item(X) or write_item(X) such that TS(T) >write_TS(X) has its read or write operation delayed until the transaction T' that wrote the value of X (hence TS(T') = write_TS(X)) has committed or aborted

To implement this algorithm, it is necessary to simulate the locking of an item X that has been written by transaction T' until T' is either committed or aborted. This algorithm does not cause deadlock, since T waits for T' only if TS(T) > TS(T').

Thomas's Write Rule. A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:

1. If read_TS(X) > TS(T), then abort and roll back T and reject the operation.
2. If write_TS(X) > TS(T), then do not execute the write operation but continue processing.
3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

10 a)Explain multi version concurrency control protocols.

Multiversion Concurrency Control Techniques

- ✓ These protocols for concurrency control keep copies of the old values of a data item when the item is updated (written); they are known as **multiversion concurrency control** because several versions (values) of an item are kept by the system. When a transaction requests to read an item, the *appropriate* version is chosen to maintain the serializability of the currently executing schedule. One reason for keeping multiple versions is that some read operations that would be rejected in other techniques can still be accepted by reading an *older version* of the item to maintain serializability.
- ✓ When a transaction writes an item, it writes a *new version* and the old version(s) of the item is retained. Some multiversion concurrency control algorithms use the concept of view serializability rather than conflict serializability.
- ✓ An obvious drawback of multiversion techniques is that more storage is needed to maintain multiple versions of the database items. In some cases, older versions can be kept in a temporary store. It is also possible that older versions may have to be maintained anyway—for example, for recovery purposes.

Multiversion Technique Based on Timestamp Ordering

- ✓ In this method, several versions $X1, X2, \ldots, Xk$ of each data item X are maintained. For *each version*, the value of version Xi and the following two timestamps associated with version Xi are kept:

1. **read_TS(Xi).** The **read timestamp** of Xi is the largest of all the timestamps of transactions that have successfully read version Xi.
2. **write_TS(Xi).** The **write timestamp** of Xi is the timestamp of the transaction that wrote the value of version Xi.

- ✓ Whenever a transaction T is allowed to execute a write_item(X) operation, a new version $Xk+1$ of item X is created, with both the write_TS($Xk+1$) and the read_TS($Xk+1$) set to TS(T). Correspondingly, when a transaction T is allowed to read the value of version Xi, the value of read_TS(Xi) is set to the larger of the current read_TS(Xi) and TS(T).

To ensure serializability, the following rules are used:

1. If transaction T issues a write_item(X) operation, and version i of X has the highest write_TS(Xi) of all versions of X that is also *less than or equal to* TS(T), and read_TS(Xi) > TS(T), then abort and roll back transaction T; otherwise, create a new version Xj of X with read_TS(Xj) = write_TS(Xj) = TS(T).
2. If transaction T issues a read_item(X) operation, find the version i of X that has the highest write_TS(Xi) of all versions of X that is also *less than or equal to* TS(T); then return the value of Xi to transaction T, and set the value of read_TS(Xi) to the larger of TS(T) and the current read_TS(Xi).

Multiversion Two-Phase Locking Using Certify Locks

- ✓ In this multiple-mode locking scheme, there are *three locking modes* for an item— read, write, and *certify*—instead of just the two modes (read, write) discussed previously. Hence, the state of LOCK(X) for an item X can be one of read-locked, write-locked, certify-locked, or unlocked.
- ✓ In the standard locking scheme, with only read and write locks, a write lock is an exclusive lock. We can describe the relationship between read and write locks in the standard scheme by means of the **lock compatibility table** shown in Figure 5.14(a).
- ✓ An entry of *Yes* means that if a transaction T holds the type of lock specified in the column header on item X and if transaction T' requests the type of lock specified in the row header on the same item X, then T' *can obtain the lock* because the locking modes are compatible. On the other hand, an entry of *No* in the table indicates that the locks are not compatible, so T' *must wait* until T *releases* the lock.
- ✓ In the standard locking scheme, once a transaction obtains a write lock on an item, no other transactions can access that item. The idea behind multiversion 2PL is to allow other transactions T' to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing *two versions* for each item X; one version, the **committed version**, must always have been written by some committed transaction.

- The second **local version** X' can be created when a transaction T acquires a write lock on X. Other transactions can continue to read the *committed version* of X while T holds the write lock. Transaction T can write the value of X' as needed, without affecting the value of the committed version X. However, once T is ready to commit, it must obtain a **certify lock** on all items that it currently holds write locks on before it can commit; this is another form of **lock upgrading**.
- The certify lock is not compatible with read locks, so the transaction may have to delay its commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks—which are exclusive locks—are acquired, the committed version X of the data item is set to the value of version X', version X' is discarded, and the certify locks are then released. The lock compatibility table for this scheme is shown in Figure 5.14(b).

(a)

	Read	Write
Read	Yes	No
Write	No	No

(b)

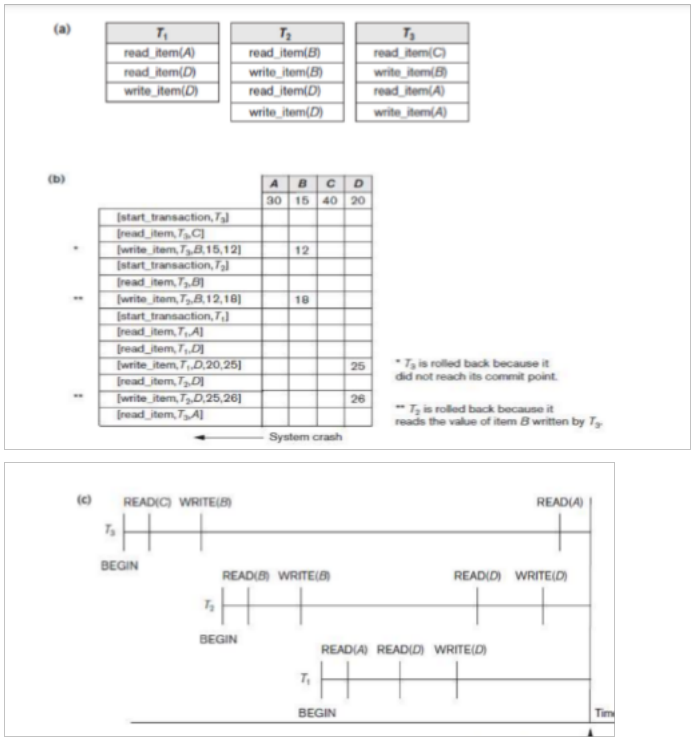
	Read	Write	Certify
Read	Yes	Yes	No
Write	Yes	No	No
Certify	No	No	No

10 b)Write short notes on the following

I. transaction rollback and cascading rollback. II. transaction support in SQL III. shadow paging IV. NO- UNDO/REDO Recovery Based on Deferred Update V. Recovery Techniques Based on Immediate Update

Transaction Rollback and Cascading Rollback)

- If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database on disk, they must be restored to their previous values (BFIMs). The undo- type log entries are used to restore the old values of data items that must be rolled back.
- If a transaction T is rolled back, any transaction S that has, in the interim, read the value of some data item X written by T must also be rolled back. Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on. This phenomenon is called cascading rollback, and it can occur when the recovery protocol ensures recoverable schedules but does not ensure strict or cascadeless schedules. Understandably, cascading rollback can be complex and time-consuming. That is why almost all recovery mechanisms are designed so that cascading rollback is never required.



Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

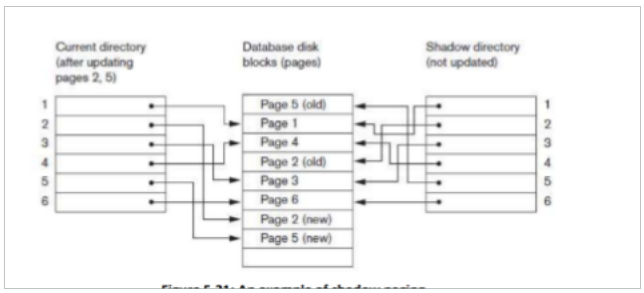
- Figure a shows an example where cascading rollback is required. The read and write operations of three individual transactions are shown in Figure (a). Figure(b) shows the system log at the point of a system crash for a particular execution schedule of these transactions. The values of data items A, B, C, and D, which are used by the transactions, are shown to the right of the system log entries. We assume that the original item values, shown in the first line, are A = 30, B = 15, C = 40, and D = 20. At the point of system failure, transaction T3 has not reached its conclusion and must be rolled back. The WRITE operations of T3, marked by a single * in Figure (b), are the T3 operations that are undone during transaction rollback. Figure (c) graphically shows the operations of the different transactions along the time axis.

Shadow Paging

- This recovery scheme does not require the use of a log in a single- user environment. In a multiuser environment, a log may be needed for the concurrency control method. ⎫
- Shadow paging considers the database to be made up of a number of fixedsize disk pages (or disk blocks)—say, n—for recovery purposes. A directory with n entries is constructed, where the ith entry points to the ith database page on disk.
- ⎫ The directory is kept in main memory if it is not too large, and all references—reads or writes—to database pages on disk go through it. When a transaction begins executing, the current directory— whose entries point to

the most recent or current database pages on disk—is copied into a shadow directory.

- ⎫ The shadow directory is then saved on disk while the current directory is used by the transaction.
- ⎫ During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten.
- ⎫ Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block.
- ⎫ To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory.
- ⎫ The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded.
- ⎫ Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NOUNDO/NO- REDO technique for recovery.
- ⎫ In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique.
- ⎫ One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. ⎫
- Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits. The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation.



NO- UNDO/REDO Recovery Based on Deferred Update

- The idea behind deferred update is to defer or postpone any actual

updates to the database on disk until the transaction completes its execution successfully and reaches its commit point.

- ⎪ During transaction execution, the updates are recorded only in the log and in the cache buffers. After the transaction reaches its commit point and the log is forcewritten to disk, the updates are recorded in the database.
- ⎪ If a transaction fails before reaching its commit point, there is no need to undo any operations because the transaction has not affected the database on disk in any way. Therefore, only REDOtype log entries are needed in the log, which include the new value (AFIM) of the item written by a write operation.
- ⎪ The UNDO- type log entries are not needed since no undoing of operations will be required during recovery. Although this may simplify the recovery process, it cannot be used in practice unless transactions are short and each transaction changes few items.
- For other types of transactions, there is the potential for running out of buffer space because transaction changes must be held in the cache buffers until the commit point, so many cache buffers will be pinned and cannot be replaced. We can state a typical deferred update protocol as follows:
  1. A transaction cannot change the database on disk until it reaches its commit point; hence all buffers that have been changed by the transaction must be pinned until the transaction commits (this corresponds to a no- steal policy).
  2. A transaction does not reach its commit point until all its REDO- type log entries are recorded in the log and the log buffer is force- written to disk.
- Notice that step 2 of this protocol is a restatement of the write- ahead logging (WAL) protocol. Because the database is never updated on disk until after the transaction commits, there is never a need to UNDO any operations.
- ⎪ REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the database on disk. In this case, the transaction operations are redone from the log entries during recovery. For multiuser systems with concurrency control, the concurrency control and recovery processes are interrelated.
- Consider a system in which concurrency control uses strict two- phase locking, so the locks on written items remain in effect until the transaction reaches its commit point. After that, the locks can be released. This ensures strict and serializable schedules.

Recovery Techniques Based on Immediate Update

- In these techniques, when a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point. Notice that it is not a requirement that every update be applied immediately to disk; it is just

possible that some updates are applied to disk before the transaction commits.

- ⎪ Provisions must be made for undoing the effect of update operations that have been applied to the database by a failed transaction. This is accomplished by rolling back the transaction and undoing the effect of the transaction's write_item operations
- . ⎪ Therefore, the UNDO- type log entries, which include the old value (BFIM) of the item, must be stored in the log. Because UNDO can be needed during recovery, these methods follow a steal strategy for deciding when updated main memory buffers can be written back to disk. T

Theoretically, we can distinguish two main categories of immediate update algorithms.

1. If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO any operations of committed transactions. This is called the UNDO/NO- REDO recovery algorithm. In this method, all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed. Hence, this method must utilize the steal/force strategy for deciding when updated main memory buffers are written back to disk.
2. If the transaction is allowed to commit before all its changes are written to the database, we have the most general case, known as the UNDO/REDO recovery algorithm. In this case, the steal/no- MODULE- 5 force strategy is applied. This is also the most complex technique, but the most commonly used in practice.

We will outline an UNDO/REDO recovery algorithm and leave it as an exercise for the reader to develop the UNDO/NO- REDO variation. When concurrent execution is permitted, the recovery process again depends on the protocols used for concurrency control.

The procedure RIU_M (Recovery using Immediate Updates for a Multiuser environment) outlines a recovery algorithm for concurrent transactions with immediate update (UNDO/REDO recovery).

Assume that the log includes checkpoints and that the concurrency control protocol produces strict schedules—as, for example, the strict two- phase locking protocol does. Recall that a strict schedule does not allow a transaction to read or write an item unless the transaction that wrote the item has committed. However, deadlocks can occur in strict two- phase locking, thus requiring abort and UNDO of transactions. For a strict schedule, UNDO of an operation requires changing the item back to its old value (BFIM).

Procedure RIU_M (UNDO/REDO with checkpoints).

1. Use two lists of transactions maintained by the system: the committed transactions since the last checkpoint and the active transactions.
2. Undo all the write_item operations of the active (uncommitted)

transactions, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.

3. Redo all the write_item operations of the committed transactions from the log, in the order in which they were written into the log, using the REDO procedure defined earlier.

The UNDO procedure is defined as follows:

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

Transaction Support in SQL

- With SQL, there is no explicit Begin_Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered. However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK. Every transaction has certain characteristics attributed to it. These characteristics are specified by a SET TRANSACTION statement in SQL.
- The characteristics are the access mode, the diagnostic area size, and the isolation level. The access mode can be specified as READ ONLY or READ WRITE. The default is READ WRITE, unless the isolation level of READ UNCOMMITTED is specified, in which case READ ONLY is assumed. A mode of READ WRITE allows select, update, insert, delete, and create commands to be executed. A mode of READ ONLY, as the name implies, is simply for data retrieval.
- The diagnostic area size option, DIAGNOSTIC SIZE n, specifies an integer value n, which indicates the number of conditions that can be held simultaneously in the diagnostic area. These conditions supply feedback information (errors or exceptions) to the user or program on the n most recently executed SQL statement.
- The isolation level option is specified using the statement ISOLATION LEVEL , where the value for can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The default isolation level is SERIALIZABLE, although some systems use READ COMMITTED as their default. I
- If a transaction executes at a lower isolation level than SERIALIZABLE, then one or more of the following three violations may occur:
- 1. Dirty read. A transaction T1 may read the update of a transaction T2, which has not yet committed. If T2 fails and is aborted, then T1 would have read a value that does not exist and is incorrect.
- 2. Nonrepeatable read. A transaction T1 may read a given value from a

table. If another transaction T2 later updates that value and T1 reads that value again, T1 will see a different value.

- 3. Phantoms. A transaction T1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE- clause. Now suppose that a transaction T2 inserts a new row r that also satisfies the WHERE- clause condition used in T1, into the table used by T1. The record r is called a phantom record because it was not there when T1 starts but is there when T1 ends. T1 may or may not see the phantom, a row that previously did not exist. If the equivalent serial order is T1 followed by T2, then the record r should not be seen; but if it is T2 followed by T1, then the phantom record should be in the result given to T1. If the system cannot ensure the correct behavior, then it does not deal with the phantom record problem.