PGCA/Sc IT/PGDCA (S2) 03
Exam. Code: FDMS

Fundamentals of Database Management Systems

SEMESTER II

MASTER OF COMPUTER APPLICATION / MASTER OF SCIENCE IN INFORMATION TECHNOLOGY / POST GRADUATE DIPLOMA IN COMPUTER APPLICATION

Block 2



KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Gauhati University
Prof. (Retd.) Pranhari Talukdar, Gauhati University
Prof. Sikhar Kumar Sarma, Cotton University
Dr. Jyotiprokash Goswami, Assam Engineering College

Course Co-ordinators

Dr. Tapashi Kashyap Das, KKHSOU
Ms. Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
9	Mr. Aniruddha Deka, Royal Global University
10	Mr. Arabinda Saikia, KKHSOU
11 & 12	Mr. Dwipen Laskar, Gauhati University
13	Dr. Uzzal Sharma, Assam Don Bosco University
14	Dr. Chandan Jyoti Kumar, Cotton University
15	Mr. Biswajit Das, Cotton University

Editorial Team

Content : Dr. Tapashi Kashyap Das, KKHSOU
Language : Prof. (Retd.) Robin Goswami, Cotton College
Structure, Format & Graphics: Dr. Tapashi Kashyap Das, KKHSOU

April, 2019 ISBN: 978-93-89123-02-9

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State University is made available under a Creative Commons Attribution-Non Commercial-ShareAlike4.0 License (International): http.//creativecommons.org/licenses/by-nc-sa/4.0.

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Headquarters : Patgaon, Rani Gate, Guwahati-781 017
City Office : Housefed Complex, Dispur, Guwahati-781 006; Web: www.kkhsou.in

The university acknowledges with thanks the financial support provided by the **Distance Education Bureau, UGC** for the preparation of this study material.

MASTER OF COMPUTER APPLICATION /
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY /
POST GRADUATE DIPLOMA IN COMPUTER APPLICATION

Fundamentals of Database Management Systems

Block-2

DETAILED SYLLABUS

		Pages
UNIT 9:	**Database Design** Design of Relational Database, Functional Dependency Diagram, Dependency-preserving property	**165-174**
UNIT 10:	**Normalization of Database** Introduction to Normalization, Anomalies of un-normalized Database, Normalization of Database: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF	**175-192**
UNIT 11:	**System Implementation Techniques** Query processing and optimization- translation between SQL queries and relational algebra	**193-214**
UNIT 12:	**Transaction Processing** Transaction and System concepts, Desirable properties, Schedules and Recoverability	**215-233**
UNIT 13:	**Concurrency Control** Locking Techniques, Concurrency Control based on timestamp ordering, Multiversion Concurrency Control Techniques	**234-257**
UNIT 14:	**Database Recovery** Concepts and techniques, Recovery in Multi Database Systems	**258-265**
UNIT 15:	**Security and Authentication** Issues, Access Control Techniques, Introduction to Multilevel Security	**266-280**

BLOCK INTRODUCTION

This is the *second block* of the course ***Fundamentals of Database Management System***. This block contains seven units.

Unit 9 covers the topic of database design. This unit primarily discusses relational database design along with the concept of functional dependency.

Unit 10 discusses the most important concept “Normalization”. All the five important normal forms are covered in details with examples in this unit.

Unit 11 is on system implementation techniques. Concept of query processing and optimization, and relational algebra are introduced in this unit.

Unit 12 is on transaction processing. Transaction processing is a very important concept of DBMS.Without transactions, most serious business applications would simply not work.

Unit 13 discusses concurrency control.

Unit 14 gives the concept of database recovery.

Unit 15 is the last unit of this course which deals with the aspects of database security and authentication.

Each unit of these blocks includes some along-side boxes to help you know some of the difficult, unseen terms. Some “EXERCISES” have been included to help you apply your own thoughts. You may find some boxes marked with: “LET US KNOW”. These boxes will provide you with some additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ANSWERS TO CHECK YOUR PROGRESS” given at the end of each unit.

UNIT 9: DATABASE DESIGN

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 Design of Relational Database

9.4 Functional Dependency

9.5 Functional Dependency Diagram

9.5.1 Rules for Functional Dependency Diagram

9.6 Let Us Sum Up

9.7 Further Reading

9.8 Answers to Check Your Progress

9.9 Model Questions

9.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the design of a relational database
- describe functional dependency, functional dependency diagram and its rules.

9.2 INTRODUCTION

In the previous unit we have learned how to normalize a database. Different normal forms are discussed in the previous unit.

This unit is on database design. We will learn various important concepts like functional dependency, rules for drawing functional dependency diagram etc.

9.3 DESIGN OF RELATIONAL DATABASE

We have already learned various terms of relational data model and concept of keys in *Unit 5* of *Block 1*. These concepts are prerequisite to learn relational database design. A relational database organizes data in tables (or relations). A table is made up of rows and columns. A row is also

called a record (or tuple). A column is also called a field (or attribute). A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data. A language called SQL (Structured Query Language) was developed to work with relational databases. A well-designed database shall:

- **Eliminate Data Redundancy:** The same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- **Ensure Data Integrity and Accuracy:** Relational database design (RDD) models information and data into a set of tables with rows and columns. Each row of a relation/table represents a record, and each column represents an attribute of data. The Structured Query Language (SQL) is used to manipulate relational databases. The design of a relational database is composed of four stages, where the data are modeled into a set of related tables. These four stages are as follows:
  i) Relations and attributes: The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities.
  ii) Primary keys: The attribute or set of attributes that help in uniquely identifying a record is identified and assigned as the primary key
  iii) Relationships: The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are: One-to-one, One-to-many, Many-to- many. An entity-relationship diagram can be used to depict the entities, their attributes and the relationship between the entities in a diagrammatic way.
  iv) Normalization: This is the process of optimizing the database structure. Normalization simplifies the database design to avoid redundancy and confusion.

By applying a set of rules, a table is normalized into the above normal forms in a linearly progressive fashion. The efficiency of the design gets better with each higher degree of normalization.

9.4 FUNCTIONAL DEPENDENCY

Functional dependency is a term derived from mathematical theory, which states that for every element in the attribute (which appears on some row), there is a unique corresponding element (on the same row). A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the *determinant* and the other attribute is called the *determined.* For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and B is the determined then we say that A functionally determines B and it is graphically represented as A→B. The following table 9.1 illustrates A→B since for each value of A there is associated one and only one value of B.

Table 9.1: A functionally determines B

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Again, the table 9.2 illustrates that A does not functionally determine B.

Table 9.2: A does not functionally determine B

A	B
1	1
2	4
3	9
4	16
3	10

Since, for A = 3, there is associated more than one value of B.

Functional dependency can also be defined as follows:

An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Let us consider the database having the following tables:

Table 9.3: Supplier table

Sno	Sname	Status	City
S1	Suneet	20	Guwahati
S2	Ankit	10	Tezpur
S3	Amit	10	Tezpur

Table 9.4: Part table

Pno	Pname	Color	Weight	City
P1	Nut	Red	12	Guwahati
P2	Bolt	Green	17	Tezpur
P3	Screw	Blue	17	Tezpur
P4	Screw	Red	14	Guwahati

Table 9.5: Shipment table

Sno	Pno	Qty
S1	P1	270
S1	P2	300
S1	P3	700
S2	P1	270
S2	P2	700
S3	P2	300

Here, in ***Supplier*** table (Table 9.3)

Sno – Supplier number of supplier that is unique

Sname – Supplier name

City – City of the supplier

Status – Status of the city e.g., A grade cities may have status 10, B grad cities may have status 20 and so on.

Here, ***Sname*** is functionally dependent on ***Sno***. Because, ***Sname*** can take only one value for the given value of ***Sno*** (e.g., S1) or in other words there must be one ***Sname*** for supplier number S1.

Functional dependency (FD) is shown by the symbol → which means that ***Sname*** is functionally dependent on ***Sno***.

Sno→Sname

Similarly, ***City*** and ***Status*** are also functionally dependent on ***Sno***, because for each value of ***Sno*** there will be only one City and Status.

Similarly, in case of ***Shipment*** table, the attributes are:

Sno – Supplier number of the supplier

Pno – Part number supplied by supplier

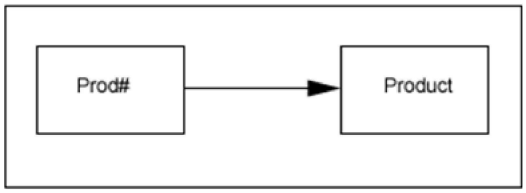
Qty – Quantity supplied by supplier for a particular Part no

In this case, **Qty** is functionally dependent on combination of ***Sno, Pno*** because each combination of **Sno** and **Pno** results only for one Quantity.

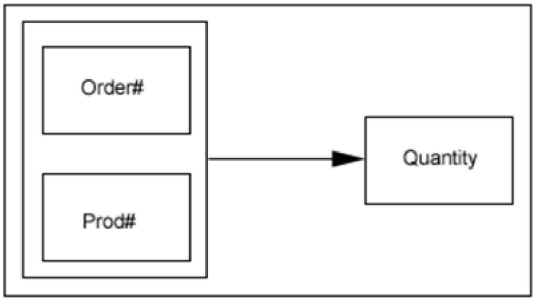
Shipment (Sno, Pno) → Shipment.QTY

9.5 FUNCTIONAL DEPENDENCY DIAGRAM

A set of Functional Dependencies for a data model can be documented in a Functional Dependency Diagram. In a Functional Dependency Diagram each attribute is shown in a rectangle with an arrow indicating the direction of the dependency. The figure below illustrates the functional dependency **Prod# > Product**.

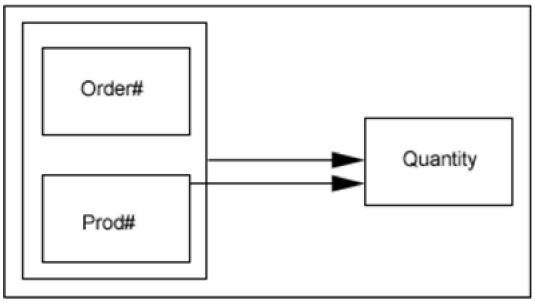


A Functional Dependency with Multiple Attributes is shown below, for the functional dependency **Order#, Prod# > Quantity**.



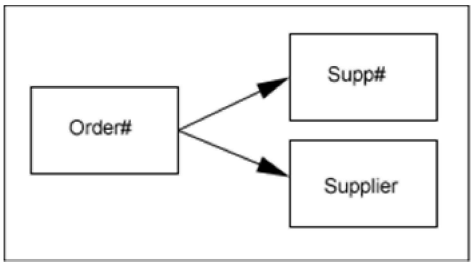
A derived Functional Dependency involving **Partial Key Dependency** is shown in the following figure.

The arrow connected to the outer rectangle, which represents **Order#, Prod# > Product** can be deleted without loss of information.



A derived Functional Dependency involving ***Transitive Dependency*** is shown in the following figure below.

The arrow which represents **Order# > Supplier** can be deleted without loss of information.

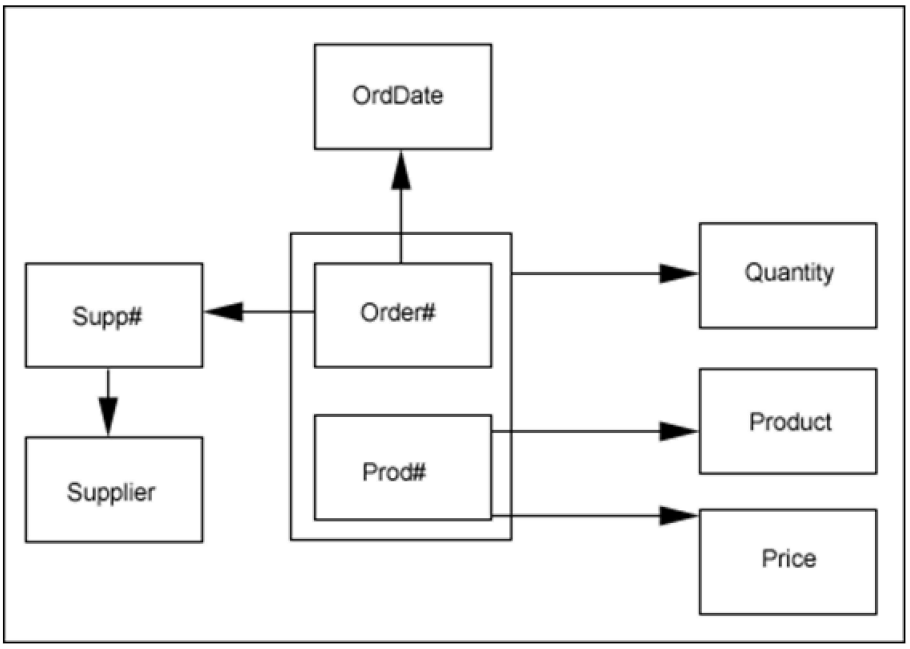


9.5.1 Rules for Functional Dependency Diagram

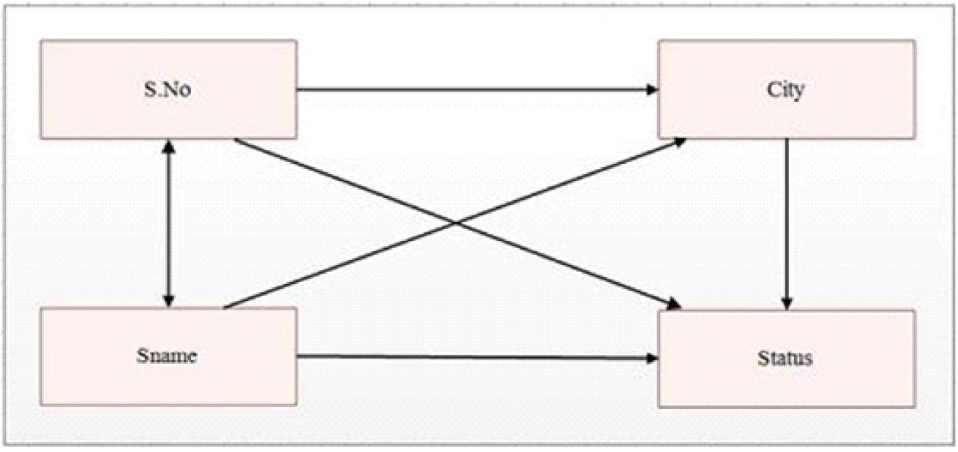
The following rules should be applied to Functional Dependency Diagrams:

- each attribute appears only once on the Functional Dependency Diagram
- all the attributes of interest appear on the Functional Dependency Diagram
- no partial key dependencies appear on the Functional Dependency Diagram
- no transitive dependencies appear on the Functional Dependency Diagram

The complete Functional Dependency Diagram for the *Purchase Order* data model is shown below:



A dependency diagram consists of the attribute names and all functional dependencies in a given table. The dependency diagram of Supplier table is:



Here, following functional dependencies exist in ***Supplier*** table:

Sno→Sname

Sname→Sno

Sno→City

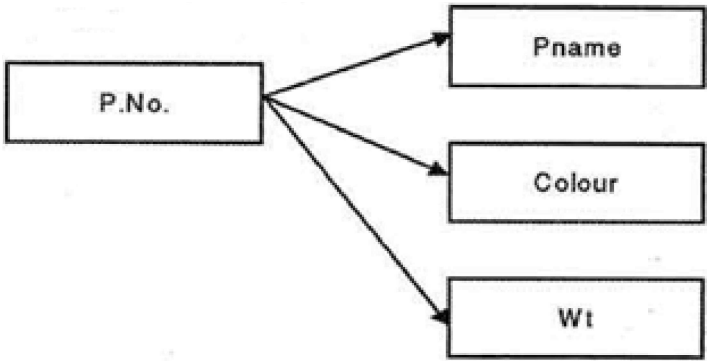
Sno→Status

Sname→City

Sname→Status

City→Status

The FD diagram of relation ***Part*** is:



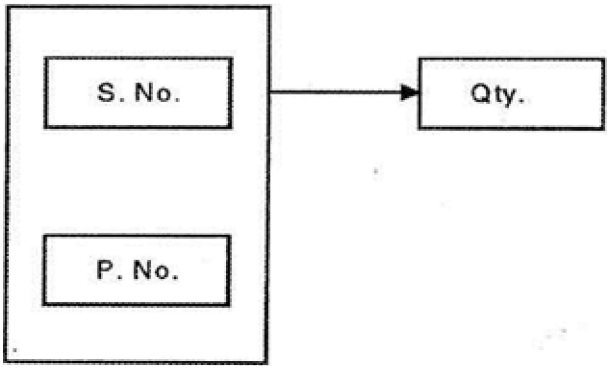
Here, following functional dependencies exist in ***Part*** table:

Pno→Pname

Pno→Color

Pno→Wt

The FD diagram of relation ***Shipment*** is:



Here, following functional dependencies exist in ***Parts*** table:

Shipment(Sno, Pno)→Shipment.QTY



CHECK YOUR PROGRESS

Q.1: Define functional dependency?

Q.2: Consider the following table:

EmpID	EmpName	Age
E101	Kaveri	27
E102	Arjun	25
E103	Bikram	29

Do you see any functional dependence between any two columns of the table?



9.6 LET US SUM UP

- A well-designed database shall eliminate data redundancy, ensure data integrity and accuracy.
- A functional dependency is an association between two attributes of the same relational database table.
- We can define functional dependency for two columns A and B of a table, as follows: Column B is said to be functionally dependent on column A if, given A we can precisely determine B.
- Dependency preservation and lossless join property are important factor for database design.



9.7 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
3) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.



9.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined.

Ans. to Q. No. 2: Let us consider a combination of the **EmpID** and the **EmpName** columns. Again consider the following two statements:

- Given an EmpID, we can certainly determine a precise EmpName

- Given an EmpName, we can certainly determine a precise EmpID.

We can observe that the first statement is true, but the second one is not. This is because, EmpID is the primary key of the table and therefore, always unique. However, EmpName can repeat and, therefore, can be a duplicate in the table. Thus, we can determine with confidence an employee's name, given her employee ID. However, the reverse is not true.

Thus, **EmpName** is functionally dependent on **EmpID**. Symbolically, this is written as **EmpID→EmpName**

But, **Age NOT→EmpID** (because more than one employee can have the same age)



9.9 MODEL QUESTIONS

Q.1: Explain the concept of functional dependency.

Q.2: What is functional dependency diagram?

Q.3: In the following column combinations, identify the functional and transitive dependencies:

a) ItemID, ItemName, ItemPrice, QuantitySold, BillAmount

b) RollNo, StudentName, Address, Subject, Marks.

*** ***** ***

UNIT 10: NORMALIZATION OF DATABASE

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 Normalization and Its Objectives

10.4 Normal Forms

10.4.1 First Normal Form (1NF)

10.4.2 Second Normal Form (2NF)

10.4.3 Third Normal Form (3NF)

10.5 Boyce-Codd Normal Form (BCNF)

10.6 Fourth Normal Form (4NF)

10.7 Fifth Normal Form (5NF)

10.8 Let Us Sum Up

10.9 Further Reading

10.10 Answers to Check Your Progress

10.11 Model Questions

10.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about normalization
- list out the objectives of normalizations
- describe first, second and third normal forms
- illustrate BCNF, fourth and fifth normal forms.

10.2 INTRODUCTION

In the previous units, we have acquainted you with Structured Query Language (SQL) and various commands of SQL with the help of which we can communicate with and retrieve data in a database. We have also discussed two important concepts in database design i.e., functional dependency and decomposition which usually help in minimizing the redundancy in database design.

In this unit we will discuss normalization which is one of the most important concept in database. Normalization is the process of efficiently organizing data in a database. Normalization is the name given to the process of simplifying the relationship among data elements in a record. In this unit we will discuss different normal forms. Learners will be able to normalize database tables into different normal forms after going through this unit.

10.3 NORMALIZATION AND ITS OBJECTIVES

Normalization is a process of decomposing a set of relations(table) with anomalies to produce smaller and well structured relations that contain minimum or no redundancy. The basic objectives of normalization are to reduce redundancy, which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Normalization provides the designer with a systematic and scientific process of grouping of attributes in a relation.

The normalization process can be defined as a procedure of analyzing and successive reduction of the given relational schemas based on their functional dependencies and primary keys to achieve the desirable properties of:

- minimizing redundancy,
- minimizing insertion, deletion and update anomalies.

The process of normalization was first proposed by ***E.F. Codd***. Normalization is a bottom up design technique for relational database. The objectives of the normalization process are as follows:

- To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- To free relations from undesirable insertion, update and deletion anomalies.
- To reduce the need for restructuring the relations as new data types are introduced.
- To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

10.4 NORMAL FORMS

Whenever the simple rules of functional dependencies are applied to a relations, it transforms the relations into a state which called *normal form.* The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. Various types of normal forms are used in relational database, they are:

- First normal form (1 NF)
- Second normal form (2 NF)
- Third normal form (3 NF)
- Boyce/Codd normal form (BCNF)
- Fourth normal form (4 NF)
- Fifth normal form (5NF)

10.4.1 First Normal Form (1NF)

A relation is said to be in first normal form if the values in the domain of each attribute of the relation are atomic. The first normal form prohibits multivalued attributes, composite attributes and their combinations. It means that 1NF disallows having a set of values, a tuple of values, or combination of both as an attribute value for a single tuple.

Let us consider the relation TRAVEL_INFO as shown in Table 10.1.

Table 10.1: Un-normalized relation TRAVEL_INFO

PERSON	VISITED_CITY	
HEMANGA	CITY	JOURNEY_DATE
	DELHI	04-07-02
	MUMBAI	15-09-03
	AGRA	02-03-04
BIKASH	CITY	JOURNEY_DATE
	CHENNAI	05-01-02
	KOLKATA	13-02-03
	PUNE	21-08-04

Here, in the relation the domain VISITED_CITY is not very simple. and in un-normalized form. Now, let us combine the respective rows in VISITED_CITY with the value of the attribue PERSON and the resultant relation will be as follows:

Table 10.2 Modified relation TRAVEL_INFO

PERSON	CITY	JOURNEY_DATE
HEMANGA	DELHI	04-07-02
HEMANGA	MUMBAI	15-09-03
HEMANGA	AGRA	02-03-04
BIKASH	CHENNAI	05-01-02
BIKASH	KOLKATA	13-02-03
BIKASH	PUNE	21-08-04

Let us consider another relation PATIENT_DOCTOR, which keeps the records of appointment details of patient to doctors. This relation is in first normal form (1NF). The relational table can be depicted as:

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, PHONE, DURATION)

Table 10.3: Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	06-02-05 16:00	15

From the relational table we have observed that a doctor cannot have two simultaneous appointments so D_NAME and DATE_TIME is a compolsite key. Similarly, a patient cannot have

same time from two different doctors. Therefore, P_NAME and DATE_TIME attributes are also a candidate key.

Problems in 1 NF: 1NF contains redundant information. In our example, PATIENT_DOCTOR relation has the following errors:

a) There exists redundant information in patients date of birth and phone number.

b) A doctor, who does not currently have an appointment with a patient, cannot be represented.

c) A patient, who does not currently have an appointment with a doctor, cannot be represented.

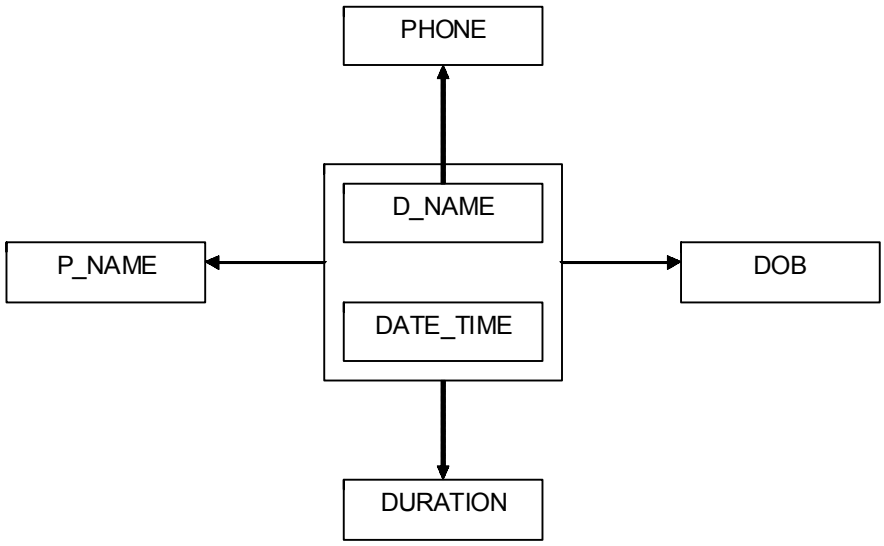


Figure 10.1: FDD of relation PATIENT_DOCTOR

Functional dependancy diagram(FDD) of the relation PATIENT_DOCTOR is shown. Here, P_NAME, DURATION, DOB are dependent on the key D_NAME & DATE_TIME and doctor's contact number i.e., PHONE is only dependent on the D_NAME.

10.4.2 Second Normal Form (2NF)

A relation or table is said to be in second normal form (2NF) if and only if–

a) it is in 1NF,

b) each non-primary key attribute is fully functionally dependent on primary key of that relation.

2NF is an intermediate step towards higher normal forms. 2NF is based on the concept of *full functional dependency*. It eliminates the problems of 1NF.

So, we come to know that, no attributes of the relation (or table) should be functionally dependent on only one part of a concatenated primary key. In our example, we have seen from the functional dependancy diagram that, PHONE is partiallly dependent only on D_NAME, for which the relation is not in 2NF.

Therefore, to bring the relation into 2NF, the information about doctor and their contact numbers have to be separated from information about patient and their appointments with doctors. Thus, the relation is decomposed into two tables, namely PATIENT_DOCTOR and DOCTOR as shown below. The relational table can be depicted as :

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, DURATION)

DOCTOR (D_NAME, PHONE)

Table 10.4: Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	PHONE	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	0361-223470	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	0361-270512	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	0361-223470	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	0361-270615	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	0361-270512	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	0361-223470	06-02-05 16:00	15

Table 10.5 Relation DOCTOR

D__NAME	PHONE
Dr Deepak	0361-223470
Dr Kalyan	0361-270512
Dr Dhiraj	0361-270615
Dr Dhiraj	0361-270615
Dr Kalyan	0361-223470
Dr Dhiraj	0361-270615
Dr Kalyan	0361-270512
Dr Deepak	0361-223470

The functional dependency diagram of the above two relations are shown below:

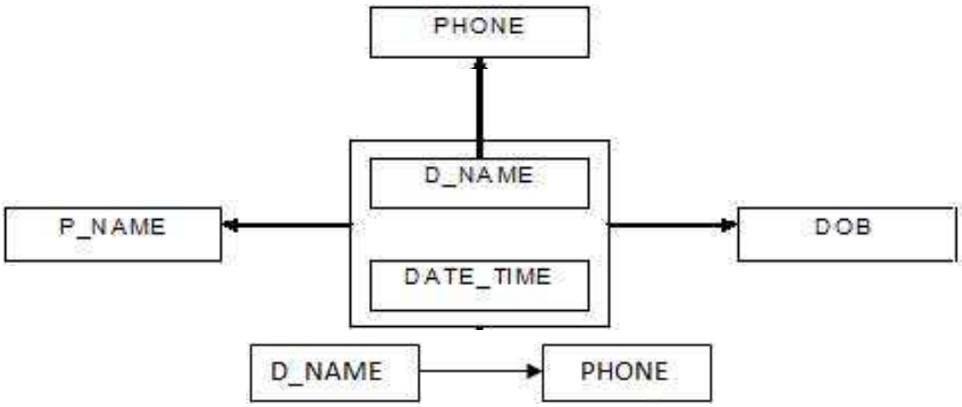


Figure 10.2: FFD of relation PATIENT_DOCTOR

Problems in 2 NF:

a) In the relation (or table) PATIENT_DOCTOR, deleting a record from it may lose patient's details.

b) Any changes made on the PATIENT_DOCTOR table may involve in changing multiple records because the information in the table is still redundant.

10.4.3 Third Normal Form (3NF)

A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are:

a) mutually independent,

b) functionally dependent on the primary key.

It means that no attributes of the relation should be transitively functionally dependent on the primary key. Thus, in 3NF, no non-prime attribue is functionally dependent on another non-prime attribute. This means that *a relation in 3NF consists of the primary key and a set of independent nonprime attributes*. 3NF is based on the problem of transitive dependency. The 3NF eliminates the problem of 2NF.

In our example, in the relation PATIENT_DOCTOR (Table 10.4), there is no dependency between the attributes P_NAME and DURATION. Again, P_NAME and DOB are mutually dependent. So, the relation is not in 3NF.

To bring the relation PATIENT_DOCTOR into 3NF, it has to be decomposed and remove the attributes that are not directly dependent on the primary key. Now, using the *transitive dependency,* DOB can be linked to the primary key, through its dependency on the P_NAME. The functional dependency diagram is as follows. Now, the new relations will be:

PATIENT_DOCTOR(P_NAME, D_NAME, DATE_TIME, DURATION)

DOCTOR (D_NAME, PHONE)

PATIENT (P_NAME, DOB)

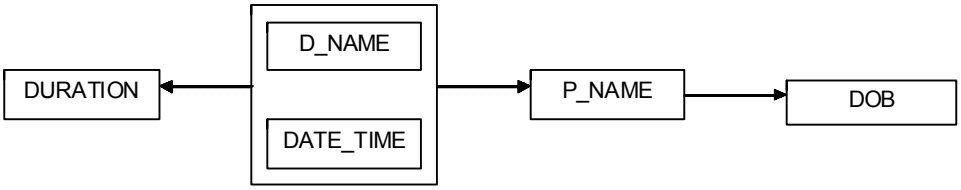


Figure 10.3: FDD for relation PATIENT_DOCTOR

Now, let us consider another relation STUDENT as shown in Table 10.6.

Table 10.6: STUDENT relation

STUDENT (Roll_No, Name, Dept, Year, Hostel_Name)

Roll_No	Name	Dept	Year	Hostel_Name
1784	Ruhit	Physics	1	Ganga
1648	Keshab	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Mrinal	Geology	3	Krishna
1485	Sumit	Zology	4	Godavari

Here, the dependency **Roll_No** $\rightarrow$ **Hostal_Name** is transitive through the following two dependencies:

Roll_No $\rightarrow$ **Year**

Year $\rightarrow$ **Hostal_Name**

Thus, the STUDENT relation is not in 3NF. To bring the relation into 3NF we can decompose the relation into two relation STUD1 and STUD2, as shown below:

STUD1 (Roll_No, Name, Dept)

STUD2 (Year, Hostel_Name)

Table 10.7: Relation STUD1

Roll_No	Name	Dept	Year
1784	Ruhit	Physics	1
1648	Keshab	Chemistry	1
1768	Gopal	Maths	2
1848	Raja	Botany	2
1682	Mrinal	Geology	3
1485	Sumit	Zology	4

Table 10.8: Relation STUD2

Year	Hostel_Name
1	Ganga
1	Ganga
2	Kaveri
2	Kaveri
3	Krishna
4	Godavari

In the above examples, the conversion into 3NF is not hard, but whenever a relation has more than one combination of attributes that may be considered as primary key then the conversion takes number of steps. Let us consider the following relation UTILIZE as shown in Table 10.9.

Table 10.9: Relation UTILIZE

Project	Proj_Manager	Machine	Qty_Used
P1	Thomas	Excavator	5
P3	John	Shovel	2
P2	Abhishek	Drilling	5
P4	Avinash	Dumper	10
P3	John	Welding	3
P1	Thomas	Drilling	4

The relation UTILIZE stores the machines information used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now, it is obvious from the table that , we can consider any one of the combination of attributes as primary key, namely, {Project, Machine} or {Proj_Manager, Machine}. The FDD for relation UTILIZE is shown figure 10.4.

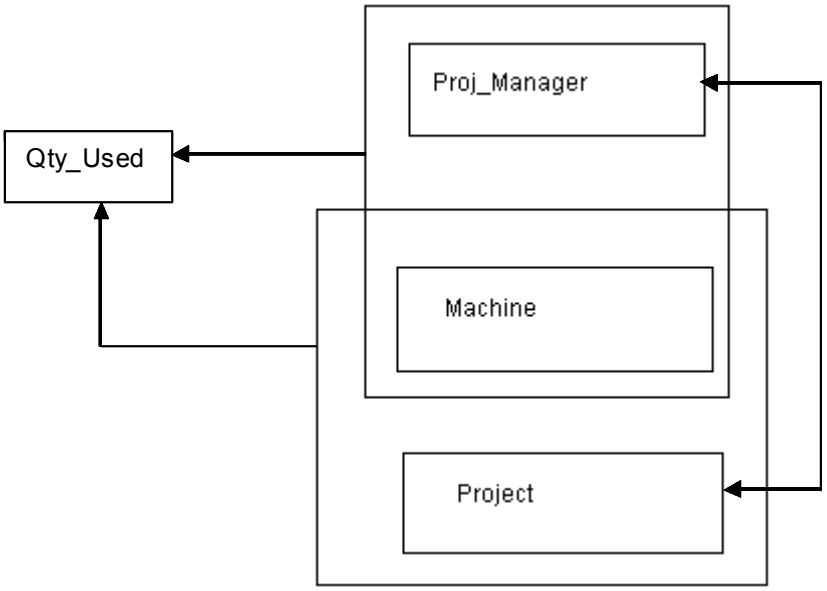


Figure 10.4: FDD of UTILIZE

In the relation UTILIZE, there is only one non-prime attribute called, **QTY_Used**, which is fully functionally dependent on each of the two relations. Thus, the relation UTILIZE is in 2NF. Moreover, there is only one non-prime attribute Qty_Used, there can be no dependencies between non-prime attributes. Thus the relation is also in 3NF.

Problems in 3NF: If we consider the above relation i.e., UTILIZE, which is in 3NF, has the following undesirable properties:

a) The project manager of each project is stored more than once.

b) A project manager cannot be stored until the project has ordered some machines.

c) A project cannot be entered unless that project's manager is known.

d) If a project's manager changes, some rows also must be changed.

10.5 BOYCE-CODD NORMAL FORM (BCNF)

The redundancy and the problems of 3NF can be eliminated by the use of the **Boyce-Codd Normal Form** (**BCNF**) which was proposed by **R. F. Boyce**.

We know that if Y is funcionally dependent on X, that is, X funcionally determines Y, then the following relation holds good:

$$X \rightarrow Y$$

Here, we call X the determinant. From this, we can define BCNF as follows:

A table is in Boyce-Codd normal form (BCNF) if the only determinants in the table are the candidate keys.

A relation (or table) R is said to be in BCNF if for every nontrivial FD : X $\rightarrow$ Y between attributes X and Y holds in R. It means:

- X is super key of R,
- X $\rightarrow$ Y is a trivial FD, that is $Y \subset X$.

From the above conditions we have come to know that, a relation must only have candidate keys as determinants. Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF.

The difference between 3NF and BCNF is that if the functional dependency $A \rightarrow B$ satisfy that B is a *primary key* attribute and A is not a candidate key, then 3NF will allow this dependency in a relation. Otherwise, if the functional dependency $A \rightarrow B$ satisfy that A must be a candidate key, then this dependency will belong to BCNF.

10.6 FOURTH NORMAL FORM (4NF)

A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

The fourth normal form is related to the concept of a ***multi-valued dependancy*** (MVD). In simple terms, if there are two columns– A and B and if for a given A, there can be multiple values of B, then we say that an MVD exists between A and B.

The fourth normal form is theoretical in nature. In practice, normalization up to the third normal form are generally adequate. In certain situations, the designers may also have to look at the BCNF.

Let us consider the following STUDENT table (or relation):

Table 10.10: STUDENT relation

Sudent	Subject	Language
Geeta	Mythology	English
Geeta	Psycology	English
Geeta	Mythology	Hindi
Geeta	Psycology	Hindi
Sikhar	Gardening	English

We can see that there are two independent MVD facts in this relationship:

a) A student can study many subjects (i.e., Student $\rightarrow \rightarrow$ *Subject)*

b) A student can learn many languages (i.e., Student $\rightarrow \rightarrow$ *Language)*

The primary key for the STUDENT table is currently a composite key made up of all the three columns in the table– *Student*, *Subject* and *Language*. i.e., here the primary key is a combination of the three fields *Student, Subject* and *Language.*

The process of bringing STUDENT table into 4NF is by spliting the independent multi-valued components of the primary key into two tables. Now, let us split these two independent multi-valued dependencies into two separate tables namely *Student_Subject* and *Student_Language*. The resulting tables are shown below:

Table 10.11: Relation Student_Subject

Student	Subject
Geeta	Mythology
Geeta	Psycology
Sikhar	Gardening

Table 10.16: Relation Student_Language

Student	Language
Geeta	English
Geeta	Hindi
Sikhar	English

We have seen that, this decomposition reduces redundancy with respect to both the independent multivalued dependency relationships, that is *Subject* and *Language*.

10.7 FIFTH NORMAL FORM (5NF)

A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.

There are some relations, which cannot be decomposed into two or higher normal form relations by means of projection methods discussed in 1NF, 2NF, 3NF and BCNF. Such relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF). The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies. Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF. 5NF is mainly used from theoretical point of view and not for practical database design.



CHECK YOUR PROGRESS

Q.1: Select the correct answer from the following:

a) Normalization is a process of–

i) decomposing of a set of relation.

ii) successive reduction of relation schema.

iii) deciding which attributes in a relation to be grouped together.

iv) all of these.

b) A normal form is–

i) a state of a relation that results from applying simple rules regarding functional dependeny.

ii) the highest normal form condition that it meets.

iii) an indication of the degree to which it has been normalized.

iv) all of these.

c) In 1NF–

i) all domains are simple.

ii) in a simple domain, all elements are atomic

iii) both (i) & (ii) iv) none of these

d) 2NF is always in–

i) 1NF ii) BCNF

iii) MVD iv) none of these

e) A relation R is said to be in 2NF,

i) if it is in 1NF.

ii) every non-prime key attributes of R is fully functionally dependent on each relation key of R.

iii) if it is in BCNF. iv) both (i) & (ii).

f) A relation R is said to be in 3NF if the,

i) relation R is in 2NF

ii) non prime attributes are mutually independent.

iii) functionally dependent on the primary key.

iv) all of these.

g) 4NF is concerned with dependencies between the elements of compound keys composed of,

i) one attributes ii) two attributes

iii) three or more attributes iv) none of these

Q.2: Fill in the blanks of the following:

a) ..... first developed the process of normalization.

b) A relation is said to be in 1NF if the values in the domain of each attribute of the relation are .....

c) 2NF can be violated only when a key is a ..... key or one that consists of more than one .....

d) In 3NF, no non-prime attribute is functionally dependent on .....

e) Relation R is said to be in BCNF if for every non-trivial FD : ..... between attributes Xand Y holds in R.

f) A relation is in BCNF if and only if every determinant is a .....

g) Any relation in BCNF is also in ..... and consequently in .....

h) 4NF is violated when a relation has undesirable .....



10.8 LET US SUM UP

- Normalization is a design technique that is widely used in designing relational model. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table).
- Functional dependencies play an important role in differencing good database design from not so good database design.
- A relation is said to be in First Normal Form (1NF) if only if every entry of the relation (the insertion of a tuple and a column) has at most a single value. In other words “a relation is in First Normal Form if only if all underlying domains contain atomic values or single value only”.
- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.
- A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are mutually independent and functionally dependent on the primary key.
- A relation R is in 4NF if and only if the following conditions are satisfied simultaneously: (i) R is already in 3NF or BCNF. (ii) If it contains no multi-valued dependencies.
- A relation (or table) R is said to be in BCNF if for every nontrivial FD : X $\rightarrow$ Y between attributes X and Y holds in R.
- A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.
- A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.



10.9 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.

2) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.

3) Ullman, J. D. (1984). *Principles of Database Systems*. Galgotia Publications.



10.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) (iv), b) (iv), c) (ii), d) (i), e) (iv), f) (iv), g) (iii)

Ans. to Q. No. 2: a) E. F. Codd, b) atomic, c) composite, attribute, d) another non prime attributes, e) X $\rightarrow$ Y, f) candidate key, g) 3NF, 2NF, h) multi-valued dependencies.



10.11 MODEL QUESTIONS

Q.1: What is meant by normalization? Why do we normalize the database? Describe the data normalization process.

Q.2: What are different normal forms?

Q.3: Define 1NF, 2NF and

Q.4: How do we convert a database in 1NF to 2NF?

Q.5: How do we convert a database in 2NF to 3NF?

Q.6: Explain 3NFwith example.

Q.7: Define BCNF. How does it differ from 3NF ? Why is BCNF stronger than 3NF? Illustrate with an example.

Q.8: Consider a relation:

Student(Student_Id, Student_Name, Student_Address, Date_of_Birth, Course_Id, Course_Name, Duration, No_of_Papers)

The functional dependencies are:

Student_Id-> Student_Name

Student_Id->Student_Address

Student_Id->Course_Id

Student_Id->Date _of_Admission

Student_Id->Date_of_Birth

Course_Id->Course_Name

Course_Id->Duration

Course_Id->No_of_Papers

Decompose the above relation into equivalent Third Normal Form (3NF) relations.

Q.9: Following relation for published books is given:

BOOK(BOO-TITLE, AUTH-NAME, BOOK-TYPE, LIST-PRICE, AUTH-AFFL, PUBLISHER)

Here, AUTH-AFFL refers to the affiliation of author. Suppose that the following FDs exists:

BOOK-TITLE $\rightarrow$ PUBLISHER, BOOK-TYPE

BOOK-TYPE $\rightarrow$ LIST-PRICE

AUTH-NAME $\rightarrow$ AUTH-AFFL

a) What normal form is the relation in ? Explain your answer.

b) Apply normalization until the relations cannot be decomposed any further. State the reason behind each decomposition.

Q.10: Explain the following terms with an example each :

a) Normalization	b) 1NF	c) BCNF
d) 3NF	e) 4NF	f) 5NF
g) MVD	h) Candidate key	i) Primary Key
j) Super Key		

*** ***** ***

UNIT 11: SYSTEM IMPLEMENTATION TECHNIQUES

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Query Processing

- 11.3.1 Basic steps in Query Processing
- 11.3.2 Translating SQL Queries into Relational Algebra
- 11.3.3 Query Tree
- 11.3.4 Measure of Query Cost

11.4 Query Algorithms

- 11.4.1 Selection Algorithms
- 11.4.2 Join Algorithms
- 11.4.3 Other Operations

11.5 Evaluation of Query Expressions

11.6 Transformation Rules for Relational Algebra Expressions

11.7 Methods of Query Optimization

11.8 Let Us Sum Up

11.9 Further Readings

11.10 Answers to Check Your Progress

11.11 Model Questions

11.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define a Query
- learn about Query Processor
- learn about different steps involved in Query Processing
- learn about parsing and translation the Query
- learn about translation between SQL queries and Relational Algebra
- learn the different query processing algorithms such as *selection*, *join* etc.

- describe the different transformation rules used for equivalent query transformation
- learn about the importance of Query Optimization and different methods of query optimization.

11.2 INTRODUCTION

We are already familiar with SQL query and Relational Algebra. We are already acquainted about different parts, components and query structures used in SQL such as DDL, DML statements, WHERE clause, Aggregation Function etc. We are also familiar with different relational algebraic expressions such as SELECT, PROJECT, DIFFERENCE, INTERSECTION, NATURAL JOIN etc. The execution and processing of query is the fundamental part of any DBMS system.

In this unit, we will learn about Query processing and Optimization of Query Processing. In the next unit, we will explore transaction processing in database system and we will learn about transaction and system concepts, desirable properties, schedules and Recoverability.

11.3 QUERY PROCESSING

Query processing is the process of transforming the high level language into low level language which can be used at the physical level of the file system, query optimization and perform the requested action for user.

11.3.1 Basics Steps in Query Processing

The fundamental steps involved in processing a query processing are *parsing and translation, optimization* and *Evaluation* as shown in figure 11.1.

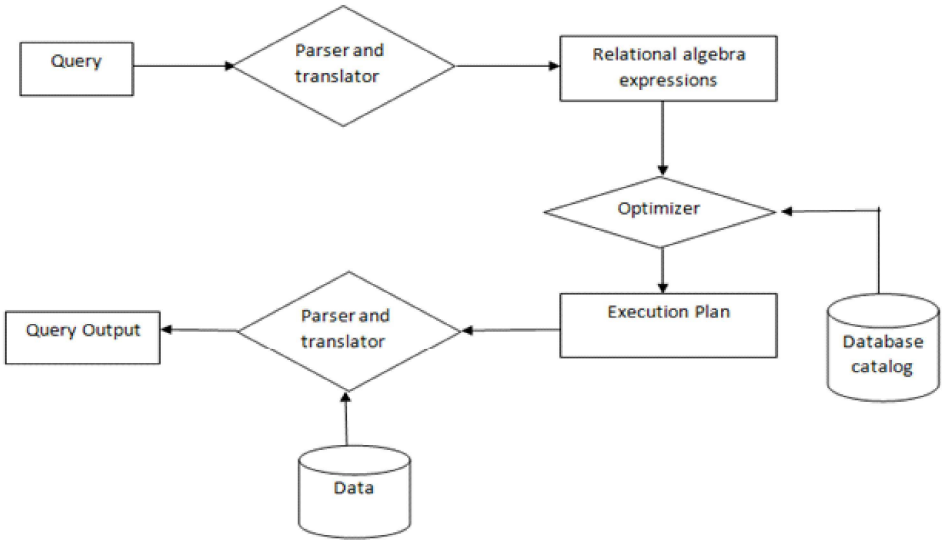


Figure 11.1: Steps in Query Processing

The *parsing and translation* is the first step of any query processing. The parser in the query processor checks the syntax of the query and validity of the relations (i.e. tables) and attributes which are used in the query. The *translator* converts each token into relational expressions, *query trees* and *query graphs*.

In the *optimization* step, the query processor applies rules to the internal data structures of the query to transform these structures into equivalent expressions (i.e. execution plan) that are more efficient to execute. The optimizer uses the statistical data i.e. information about the size of the table, the length of records, the indexes created on the table, etc. for finding out the various evaluation plans. The Optimizer also checks for the conditions and conditional attributes which are parts of the query. The different evaluation plans for a given query can have different cost in terms of time consumption to execute the query, or the Space required for executing the query. The query optimizer is responsible to find out the most efficient evaluation plan for the execution of the given query.

The final step in processing a query is the *evaluation phase*. In this phase the query evaluation engine selects the best evaluation plan generated by the query optimization engine, executes that plan and returns the answer to that query.

For example: Consider for example following SQL query:

SELECT BALANCE **FROM** ACCOUNT **WHERE** BALANCE < 2000.

This can be translated into either of the following relational algebra expressions:

- $\sigma_{BALANCE<2000}(\Pi_{BALANCE}(ACCOUNT))$
- $\Pi_{BALANCE}(\sigma_{BALANCE<2000}(ACCOUNT))$

This can also be represented as either of the following query trees:

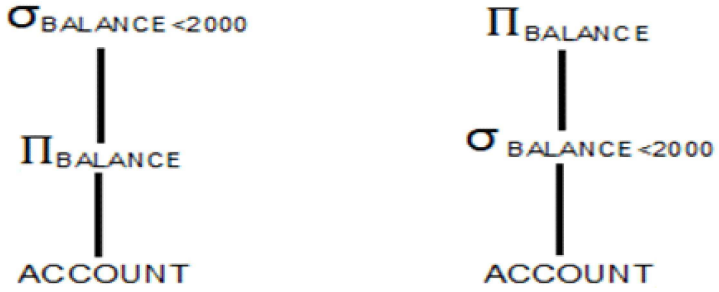


Figure 11.2: Query Evaluation Plan

The job of the query optimizer is to select the best query evaluation plan and execute it. The selection of the evaluation depends on the cost of execution of evaluation plan.

11.3.2 Translating SQL Queries into Relational Algebra

An SQL query is first translated into an equivalent relational algebra expression which is represented as a query tree data structure. The transformed data-structure is then optimized for efficient execution. An SQL query may be a composition of many sub-queries and are decomposed into *query blocks* which form the basic units. The query blocks can be translated into algebraic operators and then optimized by the query optimizer. A query block can contains a single SELECT-FROM-WHERE expression as well as clauses like GROUP BY and HAVING if these are part of the block. If there are nested queries within a query then they are identified as a separate query block.

Let us consider the EMPLOYEE relation of COMPANY relational database schema as shown in fig. 11.3.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATION

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Fig. 11.3: COMPANY relational database schema

Now, consider the following SQL query on EMPLOYEE relation in figure 11.3.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Salary > (SELECT MIN(Salary)
                FROM EMPLOYEE
                WHERE Dno=6);
```

This query includes a nested sub-query and hence would be decomposed into two query blocks: *inner block* and *outer block*. The inner block is:

(SELECT **MIN** (Salary) **FROM** EMPLOYEE **WHERE** Dno=5);

and the outer block is

(**SELECT** Fname, Lname **FROM** EMPLOYEE **WHERE** Salary>x)

Where, x represents the result returned from the inner block. The inner block could be translated into the extended relational algebra expression

$$\mathcal{F}_{\text{MAX Salary}}(\sigma_{\text{Dno=6}}(\text{EMPLOYEE}))$$

And the outer block into the expression

$\Pi_{Fname,Lname}(\sigma_{Salary>x}(EMPLOYEE))$

Now, the query optimizer chooses a query execution plan for each block. A query execution plan or a query evaluation plan is the sequence of primitive operations that can be used to evaluate a query. Query execution plan minimizes the cost of query evaluation.

11.3.3 Query Tree

A **Query tree** is a data structure that corresponds to a relational algebra expression. In this tree data-structure, the leaf nodes represent the input and the internal node represents the relational algebra operations. In query tree execution, An internal node can be executed whenever its operands are available and Node is replaced by the result of the operation it represents. The root node is executed last, and is replaced by the result of the entire tree.

A **query graph** is the data structure that corresponds to a relational calculus expression. It does not indicate an order on which to perform the operations first. There is only a single graph corresponding to each query.

Let us consider the following SQL query:

SELECT P.Pnumber, P.Dnum, E.Lname **FROM** PROJECT **AS** P, DEPARTMENT AS D, EMPLOYEE AS E **WHERE** P.Dnum = D.Dnumber AND D.Mgr_ssn = E.Ssn AND P.Plocation = 'Delhi';

This can be translated into following relational algebra expression as follows:

$\Pi_{Pnumber, Dnum, Lname}(((\sigma_{Plocation="Delhi"}(PROJECT))$

$\bowtie_{Dnum=Dnumber}(DEPARTMENT))\bowtie_{Mgr_ssn = Ssn}(EMPLOYEE))$

The above query can be represented by the Query tree as shown in figure 11.4(a) where, the three relations PROJECT, DEPARTMENT and EMPLOYEE are represented by leaf nodes P, D and E, while the relational algebra operations of the expressions are represented by internal tree nodes. When this query is executed,

the nodes marked (1) must be executed before the node marked (2) because some resulting tuples of operation (1) must be available to the operation (2) to start with. Similarly, node (2) must begin executing and producing results before node (3) can start execution.

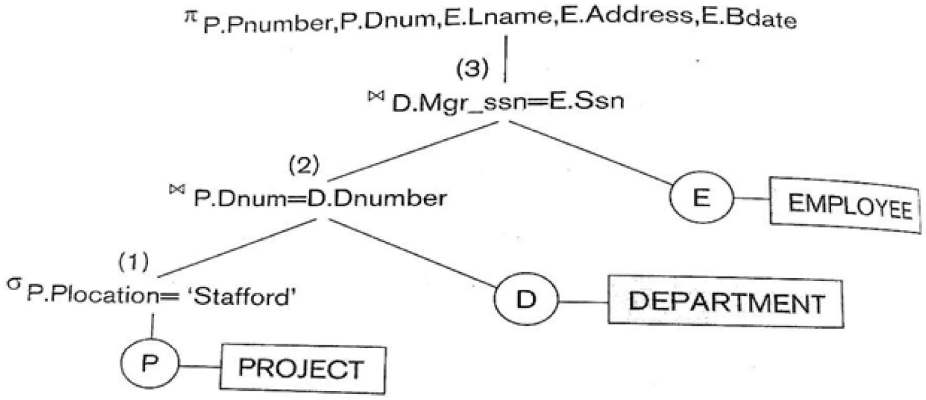


Figure 11.4 (a): Query Tree

The query graph representation of the above query is:

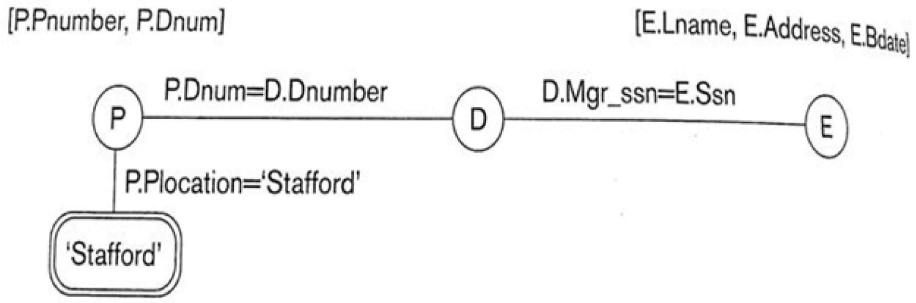


Figure 11.4 (b): Query Graph representation of the above expression

EXERCISE 11.1



Draw the Query tree for the following Relational algebra expression:

SELECT Lname **FROM** EMPLOYEE, WORKS_ON, PROJECT **WHERE** P_name='Aquarius' **AND** P_number=Pno **AND** Essn=Ssn **AND** Birth_date>'1957-12-31'

11.3.4 Measure of Query Cost

Cost of query is the time taken by the query to hit the database and to return the result. It involves query processing time i.e.; time taken to parse and translate the query, optimize it, evaluate, execute

and return the result to the user is called cost of the query. Many factors can contribute to time cost such as disk accesses, CPU or even network communication. Therefore, the cost of query evaluation can be measured in terms of following:

- *Access cost to secondary storage:* This is the cost of transferring i.e., reading and writing) data blocks between secondary disk storage and main memory buffers. The cost of searching for records in a disk file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addition, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.
- *Disk storage cost:* This is the cost of storing on disk any intermediate files that are generated by an execution strategy for the query.
- *Computation cost:* This is the cost of performing in-memory operations on the records within the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values. This is also known as CPU (central processing unit) cost.
- *Memory usage cost:* This is the cost pertaining to the number of main memory buffers needed during query execution.
- *Cost of communication (in distributed or parallel database system):* This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated. In distributed databases, it would also include the cost of transferring tables and results among various computers during query evaluation.

In large database systems, however, disk accesses are usually the most important cost, since disk accesses are slow compared to in-memory operations. Also CPU speeds have been improving much faster than have disk speeds. Thus, it is likely that the time spent in

disk activity will continue to dominate the total time to execute a query. Finally, estimating the CPU time is relatively hard compared to estimating the disk-access cost. Therefore, most people consider the disk-access cost a reasonable measure of the cost of a query-evaluation plan.

11.4 QUERY ALGORITHMS

To query a table, there are different ways to query it to get the same result. The query execution engine can have a large number of specialized algorithms designed to process particular relational operation and access path combinations. Let us discuss some examples of algorithms for various operations.

11.4.1 Selection Algorithms

The SELECT operation can be performed in a number of ways. The SELECT operation scans the records of a file to search for and retrieve records that satisfy the selection criteria. The following are some examples of simple selection algorithms.

File Scan: A number of selection algorithms are possible for selecting records from a file. These algorithms are also known as **file scans**, because they scan the records of a file to search for and retrieve records that satisfy a selection condition. Two basic file scan algorithms for selection operation are:

- **S1: Linear Search (brute force):** In linear search, the system scans each file blocks and test each record from the file is to test whether it satisfy the selection criteria. This is sometimes referred to as a *full table scans*.

  Cost of the algorithm (in terms of block transfer) is defined as:

  a) Cost of searching records satisfying a condition=Number of blocks in a database = N_b

  b) Cost of searching a key attribute value=average number of block transfer for locating the value (on average only half of

the blocks would be searched before the specific record is found) = $N_b/2$

- **S2: Binary search on primary key:** it can be used, if the selection condition involves an equality comparison on a key attribute on which the file is ordered. It is more efficient than linear search. Cost of the algorithm: Cost of finding the first tuple using the binary search method + sequence of other blocks that continue to satisfy the condition.

$$\lceil \log_2(N_b) \rceil + \frac{s}{bfr} - 1$$

Where *s* is the average number of tuples with same value and *bfr* blocking factor (number of tuples in a block) of the relation)

Index Scan: This method of selection uses indexes to traverse the record. The search key should be indexed and should be used in the filter condition for an Index scan to work.

- **S3: a) Primary index-scan for equality:** if the selection condition involves an equality comparison on a key attribute with a primary index. e.g.,

  $\sigma_{SSN='123456789'}$(EMPLOYEE),

  where *ssn* a key attribute for the EMPLOYEE. The cost of this method is:

  Cost = (number of index level) + 1 (block of the primary index is traverse to access)

  b) Hash key: it retrieves a single a single record in a direct way. The cost of this method is: Cost = 1

- **S4: Using an ordering index to retrieve multiple records:** if the comparison condition is >, >=, < or <= on a key field with an ordering index, then we need to locate the first record satisfying the condition after which the records are scanned forward or backward as the condition may be, displaying all records. Assuming uniform distribution, then we would expect half the records to satisfy the condition. The cost of this method is:

  Cost = number of index level + $N_b/2$

- **S5: Using a clustering (secondary) index to retrieve multiple records:** if the selection condition involves an equality comparison on a **non-key attribute** with a clustering index. The cost of this method is:

  Cost = number of index level + $\left\lceil \frac{s}{bfr} \right\rceil$

- **S6: Using a secondary (B+ tree) index:**
  a) **On equality condition:** on an equality comparison condition, *s* tuples will satisfy the condition on the indexing attribute. Since, the index is non-clustering, we have to assume that the tuples may reside on different blocks. The estimated cost is: Cost = number of index level + *s*
  b) **On inequality condition:** if the comparison condition is >, >=, <, or <= and assuming uniform distribution, we would expect half the file records to satisfy the condition, then the first level index blocks are accessed, plus the file records via index. The estimated cost of this method is:

  Cost=number of index level + $\left\lceil \frac{b_{I1}}{2} \right\rceil + \left\lceil \frac{r}{2} \right\rceil$

Selection Algorithms for Complex Selection:

Conjunction: Conjunction is basically a set of **AND** operations. e.g. $\sigma_{\text{DNO=5 AND Salary>3000 AND Sex='F'}}$(EMPLOYEE). The DBMS can use following additional methods to implement the operations:

- **S7: Conjunctive Selection using an individual index**: Select any algorithm from S1 to S6 on one or more conditions. If possible, test other conditions on these tuples after fetching them into memory buffer.
- **S8: Conjunctive selection using a composite index**: Use appropriate composite (multiple-key) index if available.
- **S9: Conjunctive selection by intersection of identifies:** Requires indices with record pointers. Use corresponding index for each condition, and take intersection of all the obtained sets

of record pointers. Then fetch records from file if, some conditions do not have appropriate indices, test them in memory after fetching from memory buffer.

Disjunction: Disjunction is basically a set of **OR** conditions. e.g., $\sigma_{DNO=5\ \textbf{OR}\ Salary>3000\ \textbf{OR}\ Sex='F'}(EMPLOYEE)$. *Disjunctive Selection Conditions* are much harder to process and optimize because the records satisfying the disjunctive condition are the union of the records satisfying the individual conditions.

- **S10: Disjunctive selection by union of identifies** is applicable if *all* conditions have available indices, otherwise use linear scan. Use corresponding index for each condition, and take union of all the obtained sets of record pointers, eliminate duplicates, then fetch records from file.



CHECK YOUR PROGRESS

Q.1: Suppose that a *EMPLOYEE* relation as shown in figure 11.3. has $r_E = 10{,}000$ records stored in $b_E =$ 2000 disk blocks with blocking factor $bfr_E = 5$ records/block and following statistics in the system catalog:

- A secondary index on the key attribute *Ssn*, with $X_{Ssn} = 4 (S_{Ssn} = 1)$
- A secondary index on the non-key attribute *Dno* = 2, with $X_{Dno} = 2$ and first-level index blocks $b_{I1Dno} = 4$. There are $d_{Dno} = 125$ distinct values for *Dno,* so the selection cardinality of Dno is $S_{Dno} = (r_E / d_{Dno}) = 5000$

Find out the estimated cost for following operations:

OP1: $\sigma_{Ssn='123456789'}(EMPLOYEE)$

OP2: $\sigma_{Dno>5}(EMPLOYEE)$

OP3: $\sigma_{Dno=5}(EMPLOYEE)$

11.4.2 Join Algorithms

Like selection, the JOIN operation can be implemented in a variety of ways. In terms of disk accesses, the JOIN operations can be very expensive; so implementing and utilizing efficient join algorithms is critical in minimizing a query's execution time. The following are 4 well-known types of join algorithms:

The join operation is of the form: $R \bowtie_{A=B} S$

J1: Nested-Loop Join (brute force): This algorithm consists of an inner for loop nested within an outer for loop. For each record *t* in *R* (outer for loop) retrieve every record *s* from *S* (inner for loop) and tests whether the two records satisfy the join condition t[R]=s[S]. The estimated cost of this approach is: $N_r + (N_r * N_s)$. Where, N_r is the number of block of R has to be read and N_s is the number of block of S has to be read for each block of R. The second term is fixed, but the first term could vary depending on the relation chosen for the outer loop.

J2: Index Nested-Loop Join: if there is an index (or hash function) on the join attributes of the inner relation, then we can replace the inefficient file scan with an index lookup. For each tuple in R, we use the index to retrieve the matching tuple S. The cost of scanning R is N_r, as before. However, the cost of retrieving the matching tuple is S depends on the type of index and number of matching tuples. Let, c be the cost for the lookup, then the worst -case cost for joining rand sis $N_r + n_r * c$, where, n_r is the number of tuples of R. If the join attribute A is the primary key, then c= number of index level + 1.

J3: Sort-Merge Join: For Equijoins, the most efficient join is achieved when both the relations R and S are physically sorted on the join attributes. Both relations R and S are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If R and S are not sorted then, they should be sorted by using external sorting. The estimated cost of this approach is:

$N_r + N_s$ (if already sorted) or ($N_r + N_s + N_r * \log_2 N_r + N_s * \log_2 N_s$)

J4: Hash Join: The records of files *R* and *S* are both hashed to the same hash file, using the same hashing function on the join attributes *A* of R and B of *S* as hash keys. A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets known as *partitioning phase.* A single pass through the other file (S) then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from R in that bucket known as *probing phase*. The estimate cost of this approach is: $3 * (N_r + N_s)$

11.4.1 Other Operations

There are many other operations that are performed in database systems such as *Duplicate Elimination*, *Projection*, *Aggregate Function operation*, *SET operation.*

Duplicate Elimination: There are two methods for the elimination of duplicate tuples: *Sorting* and *Hashing*. In *Sorting*, duplicates in a relation R will be adjacent to each other and may be identified easily and deleted. The estimated cost of this approach is: $N_r + N_r * \log_2 N_r$.

Another optimized method for duplicate elimination is using *Hashing* where, deletion of duplicates are done during generation as well as at intermediate merge steps in external sort-merge. Here, duplicates will be clubbed together in the same bucket and therefore may be deleted easily.

Projection: A PROJECT operation $\Pi_{<\text{attribute lits}>}(R)$ defines a relation S containing vertical subset of a relation R extracting the values of specified attributes and eliminating duplicates. Therefore, it requires two steps:

1) Removal of attributes that are not required
2) Duplicate elimination of tuples (if any)

Aggregate Function Operation: Aggregate functions (MIN, MAX, AVG, COUNT, SUM) can be implemented in a manner similar to duplicate elimination. Sorting or hashing can be used to bring tuples

in the same group together, and then aggregate functions can be applied to each group.

SET Operations: the binary SET operations Union (∪), Intersection (∩), and Set-Difference (–) can be applied to relations that are union-compatible. We can implement these operations by first sorting both relations R and S on the same attributes and then scanning through each of the sorted relations once to obtain the desired result. We can also use hashing algorithm to implement these operations.

In Union operation, $\cup_r = \{t_r \mid t_r \in R \text{ or } t_r \in S\}$

In Intersection operation, $\cap_r = \{t_r \mid t_r \in R \text{ and } t_r \in S\}$

In Set-difference operation, SD_r = {and}



CHECK YOUR PROGRESS

Q.2: Choose the appropriate option:

i) The files from which selection algorithms are selected for records are called:

a) Scan file b) Spanned file

c) Non dense file d) Indexed file

ii) The procedure of choosing a suitable query out of all the queries is classified as:

a) Query optimization b) Parser optimization

c) Processor optimization d) All of the above

iii) The representation of the query in the form of data structure is classified as:

a) Query graph b) Query tree

c) Scanner tree d) Parser tree

iv) The representation of the queries in the form of data structure graph is classified as:

a) Query graph b) Query tree

c) Scanner tree d) Parser tree

11.5 EVALUATION OF QUERY EXPRESSIONS

In the previous section we have seen the algorithm for individual operations. Now, let us discuss the methods for evaluating an entire expression. The obvious way to evaluate an expression is simply to evaluate one operation at a time, in an appropriate order. The result of each evaluation is *materialized* in a temporary relation for subsequent use. A disadvantage to this approach is the need to construct the temporary relations, which (unless they are small) must be written to disk. An alternative approach is to evaluate several operations simultaneously in a *pipeline*, with the results of one operation passed on to the next, without the need to store a temporary relation. In general there are two general methods:

- Materialization
- Pipelining

- **Materialization:** In materialization approach, each operation in the expression is evaluated one by one in appropriate order starting from the lowest-level operations in the expression (at the bottom of the tree). The result of each operation is materialized (created) in a temporary relation which becomes input to the subsequent operations. Let us consider the query tree in fig 11.4(a). In this example, we start at the selection operation on *Project* and a temporary relation is created. At the next level, this temporary relation is taken as input and join operation is performed with *Department* relations and output is stored in a temporary relation. By repeating this process, we get the ûnal result by executing the projection operation at the root of the tree, using as input the temporary relations created by the join. Materialised evaluation is always possible though; the cost of writing/ reading results to/from disk can be high if system has limited buffer size. This problem can be tackled by using *double buffer* (using two buffers, with one continuing execution of the algorithm while the other is being written out). Double buffering allows the algorithm to execute more quickly by performing CPU activity in parallel with I/O activity.

- **Pipelining:** the query-evaluation can be improved by reducing the number of temporary ûles that are produced. This is achieved this combining several relational operations into a pipeline of operations, in which the results of one operation are passed along to the next operation in the pipeline. Evaluation as just described is called pipelined evaluation. Combining operations into a pipeline eliminates the cost of reading and writing temporary relations. For example, consider the expression $\Pi_{A1,A2}(R \bowtie S)$. When the join operation generates a tuple of its result, it passes that tuple immediately to the project operation for processing instead of creating temporary relations to hold the result of the join, and then reading back in the result to perform the projection. By combining the join and the projection, we avoid creating the intermediate result, and instead create the ûnal result directly.

11.6 TRANSFORMATION RULES FOR RELATIONAL ALGEBRA EXPRESSIONS

By applying transformation rules, the query optimizer can transform one relational algebra expression into an equivalent expression that is known to be more efficient. Two relational algebraic expressions are said to be equivalent if on every legal database instance the two expressions generate the same set of tuples (order of tuples is irrelevant). Following are some of the transformation Rules for Relational Algebraic expressions:

- Cascade of σ : A select with conjunctive conditions on the attribute list is equivalent to a cascade of selects upon selects:

$$\sigma_{A_1 \wedge A_2 \ldots\ldots \wedge A_n}(R) \equiv \sigma_{A_1}(\sigma_{A2}(\ldots(\sigma_{A_n}(R))\ldots))$$

- Commutativity of σ : The select operation is commutative:

$$\sigma_{A_1}(\sigma_{A_2}(R)) \equiv \sigma_{A_2}(\sigma_{A_1}(R))$$

- Cascade of Π : A cascade of project operations is equivalent to the last project operation of the cascade:

$$\Pi_{C_1}(\Pi_{C_2} \ldots(\Pi_{C_n}(R))\ldots) \equiv \Pi_{C_1}(R)$$

- Cascade of σ with Π : If the selection condition *c* involves only those attributes $A_1, A_2, ..., ..., ..., A_n$ in the projection list, the two operations can be commuted:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_c(R)) \equiv \sigma_c(\Pi_{A_1, A_2, \ldots, A_n}(R))$$

- Commutativity of ⋈ with ×: The ⋈ operation is commutative, as is the × operation.

$$R \bowtie_c S \equiv S_c$$
$$R \times S \equiv S \times R$$

- Commutating σ with ⋈ or ×: The Select can be commuted with join (or Cartesian product) as follows.

$$R \bowtie_c S \equiv S_c$$
$$R \times S \equiv S \times R$$

  a) If all of the attributes in the select's condition are in relation R then,

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S)$$

  b) Given select the condition *c* composed of conditions c_1 and c_2, and c_1 contains only attributes from R, and c_2 contains only attributes from S, then

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$$

- Commutativity of set operations (∩, ∪, –): Union and intersection operations are commutative; but the difference operation is not:

$$R \cup S \equiv S \cup R,\ R \cap S \equiv S \cap R,\ R - S \neq S - R$$

- Associativity of (⋈, ×, ∩, ∪, –): All four of these operations are individually associative. Let θ be any of these operations, then:

$$(R\ \theta S)\ \theta T \equiv R\ \theta (S\ \theta T)$$

- Commutativity of σ with set operations (∩, ∪, –): Let, θ be any one of the three set operations then:

$$\sigma_c(R\ \theta S)\ \theta T \equiv (\sigma_c(R))\ \theta (\sigma_c(S))$$

- The projection operation distributes over the union operation:

$$\Pi_C(E_1 \cup E_2) \equiv (\Pi_C(E_1)) \cup (\Pi_C(E_2))$$

Using these theorems, an algorithm can be defined to transform the original query expression/tree created by the parser into a more optimized query.



CHECK YOUR PROGRESS

Q.3: Choose the appropriate option:

i) Each relational-algebra expression represents a particular sequence of–

a) Parameters b) Differentiation

c) Operations d) Routines

ii) The operations of theta-join are:

a) Distinctive b) Conjunctive

c) Commutative d) Associative

iii) Which of the following are steps in query processing?

a) Parsing and translation b) Optimization

c) Evaluation d) All of the mentioned

iv) A sequence of primitive operations that can be used to evaluate a query are called as .....

a) Query evaluation algebra

b) Query evaluation plan

c) Query evaluation primitive

11.7 METHODS OF QUERY OPTIMIZATION

Query optimization is the process of selecting an efficient execution plan for evaluating the query. There are two methods of query optimization:

- Cost based Query Optimization
- Heuristical Query Optimization
- **Cost based Query Optimization**: This is based on the cost of the query. A cost-based query optimizer works as follows: First, it generates all possible query execution plans. Next, the cost of each plan is estimated. Finally, based on the estimation, the plan with the lowest estimated cost is chosen. The query can use different paths

based on indexes, constraints, sorting methods etc. This method mainly uses the statistics like record size, number of records, number of records per block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc. Since the decision is made using estimated cost values, the plan chosen may actually not be optimal.

- **Heuristic Based Optimization:** Heuristic based optimization uses transformation rules to convert one relational algebra expression into an equivalent form that is known to be more efficient. It is also known as rule-based optimization approaches. The scanner and parser generates *initial query tree* a without doing any optimization. In heuristic optimization, query optimizer generates the *final query tree* by using heuristics rules. Heuristics are based on rules for equivalence among relational algebra expressions.

  Some of the common heuristic rules are:

- Perform SELECT and PROJECT operations before JOIN operations. This is done by moving the SELECT and PROJECT operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive SELECT/PROJECT operations at first before the other operations.
- Replace CARTESIAN PRODUCT and a subsequent SELECTION operation with JOIN operation.



11.8 LET US SUM UP

- Query processing and optimization is the most crucial phase a DBMS system.
- The high level query is translated to a low level operation by the query processor.
- Given a query, there are generally a variety of methods for computing the answer. It is the job or responsibility of the query optimizer to translate the query to an equivalent query in internal form that can be computed more efficiently.

- Selection of appropriate evaluation plan is impotent for query optimization.
- There are various searching algorithms available for selection operations.
- There are also various types of possible join algorithms available for choosing proper evaluation plan for join operations.
- A lot of transformation rules help the query optimizer to translate the relational query operations into equivalent expressions with optimizations.
- There are two methods of query optimization: *Cost-based approach* and *Heuristical approach*.



11.9 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.
2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



11.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: The estimated cost for ***OP1***:

Cost = b_E = 2000 (using *S1a*)

Cost = $b_E/2$ =2000/2 = 1000 (*S1b*)

Cost = X_{Ssn}+1 = 4+1 = 5 (using *S6a*)

The estimated cost for ***OP2***:

Cost=b_E=2000 (using *S1a*)

Cost=X_{Dno}+ b_{I1Dno} +$r_E/2$=2+4/2+(10000.2)=5004 (using *S6b*)

The estimated cost for ***OP3***:

Cost=b_E=2000 (using *S1a*)

Cost=X_{Dno}+ S_{Dno} =2+80=82 (using *S6a*)

Ans. to Q. No. 2: i) (a), ii) (d), iii) (b), iv) (a)

Ans. to Q. No. 3: i) (c), ii) (c), iii) (d), iv) (b)



11.11 MODEL QUESTIONS

Q.1: Define Query optimization and explain its significance for DBMS.

Q.2: Explain the different steps followed in query processing.

Q.3: Explain the different steps followed in query Optimization.

Q.4: What is a relational query tree?

Q.5: What is query evaluation plan?

Q.6: What are the different algorithms available for implementation of selection operation?

Q.7: What are the different algorithms available for implementation of Join operation?

Q.8: Discuss, how the SQL query is converted into relational algebra queries before optimization is done?

Q.9: Discuss the different types of transformation Rules for finding equivalent expressions.

Q.10: Discuss the different methods of Query Optimization.

Q.11: Explain briefly how query expressions are evaluated.

*** ***** ***

UNIT 12: TRANSACTION PROCESSING

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Concept of Transaction

- 12.3.1 A Basic Transaction Model
- 12.3.2 Single-User versus Multi-User Systems
- 12.3.3 Need of Concurrency Control
- 12.3.4 States of a Transaction
- 12.3.5 Desirable properties of a Transaction
- 12.3.6 The System Log and Commit Point of a Transaction

12.4 Schedules of Transactions

- 12.4.1 Recoverability of Schedules
- 12.4.2 Serializability of Schedules

12.5 Let Us Sum Up

12.7 Further Reading

12.6 Answers to Check Your Progress

12.8 Model Questions

12.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define a transaction
- learn about transaction processing of DBMS
- define different primitive operations of a transaction
- describe ACID properties of transactions processing
- differentiate between Single-user and multiuser systems
- define the Lost-update problem
- define the schedule of a transaction
- learn the concept of view-serializable schedule and conflict serializable
- define recoverable and cascadeless schedule
- learn about the concepts of recovery management.

12.2 INTRODUCTION

We are already familiar with Query processing and optimization. We are also acquainted with different steps and query processing and how query optimization is done internally by the system in a DBMS.

In this unit we will learn about Transaction processing where we will learn about transaction and system concepts, desirable properties, schedules and Recoverability. In the next unit we will discuss concurrency control in database system and will learn about how to ensure the serializability of transactions in a multi-user database management system.

12.3 CONCEPT OF TRANSACTION

A database transaction is a set of operations that must be processed as one unit of work. The transaction may be an entire program or a single command and may involve any number of operations on the database. A transaction is bounded by **begin transaction** and **end transaction** statement. All the operations of a transaction lie between the two above mentioned statements. A simple application program may contain several transaction boundaries. A transaction is a known as **read-only transaction** if operations in the transaction do not update the database but only retrieve data. A transaction can contain the following two basic database access operations:

- **read (X):** reads a database item named X into a program variable which is also named as X for simplicity (say).
- **write (X)**: writes the program variable named X into the database item named X.

12.3.1 A Basic Transaction Model

Consider the following transaction T consisting of T_1 and T_2. Let, T transfers an amount of £100 from account X to account Y. This transaction can be defined as below:

Begin_Transaction T1
Open_Account(X)
X.Balance = X.Balance – 100
Close_Account(X)
End transaction

Begin_Transaction T2
Open_Account(Y)
Y.Balance = Y.Balance + 100
Close_Account(Y)
End transaction

By, using the simple database access operations, the transaction model for the above transaction can be represented as follows:

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

Figure 12.1: Example of a transaction T.

12.3.2 Single User versus Multiuser Systems

A database system can be classified as single user or multi user based on number of user which uses the system. A database system is said to be **single-user** if at most one user at a time can use it and it is called as **multi-user** if more than one user can use the system at a time, i.e. the system can be used concurrently. Single-user system is restricted to personal computer system. A railway reservation system is a multi-user system. Multi-user system is possible due to the concept of **multiprogramming**. Multiple programs can run simultaneously in a multiprogramming system.

12.3.3 Need of Concurrency Control

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. Concurrency Control is the process of managing simultaneous operations on the database without having them interfere with one another and deals with interleaved execution of more than one transaction.

Following are the problems that can occur when concurrent transactions execute in an uncontrolled manner.

The lost update problem: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. Hence, the problem occurs when one user's update overwrites another user's update.

Let us consider two transactions T_1 and T_2 and suppose their operations are interleaved as shown in figure 12.2. The final value of item X is incorrect because T_2 reads the value of X before T_1 changes it in the database, and hence the updated value resulting from T_1 is lost. For example, let X = 200, Y = 300 at the start. The aim of the transaction is to transfer 100 from X to Y and then add 50 more to X. The correct result after the transaction is X = 150, Y = 400. But, in the interleaving operation shown in the fig. 12.2., it is X = 250, Y = 400, because the update in T_1 that transfers the 100 from X was *lost*.

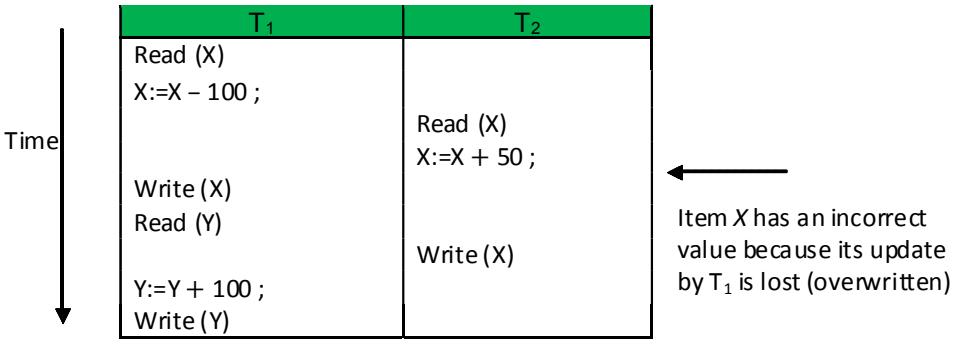


Figure 12.2: Lost Update Problem

The Temporary Update (Dirty Read) Problem: This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. This happens when one transaction can see intermediate results of another transaction before it has committed.

For example, let X = 200, Y = 300 at the start. Say, the transaction T_1 updates X by reducing 100 but fails before completion. So, the

system must change X to original state i.e. X = 200. But, before it can do so, however, transaction T_2 reads the *temporary* value of X, i.e. X = 100., which is not recorded permanently due to the failure of transaction T_1. The value of X that is read by T_2 is known as *dirty data* and the problem is known as *temporary update problem*.

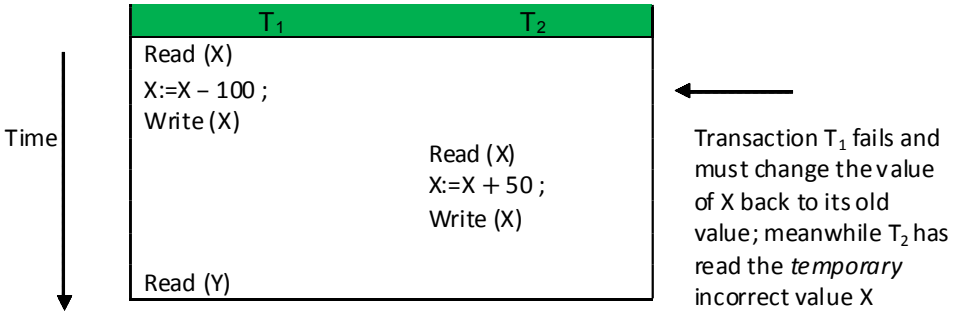


Figure 12.3: Temporary Update (Dirty Read Problem)

The Incorrect Summary Problem: This problem is caused when one of the transactions is executing an aggregate operation on several data items, and other transactions are updating one or more of those data items. This causes an inconsistent database state and is known as Inconsistent analysis or incorrect summary problem.

For example, Let T_1 and T_2 are two transaction in which transaction T_1 transfers money from account A to account B and in the mean time, transaction T_2 calculates the sum of 3 accounts namely, A, B, and C. Let A = 500, B = 500, C = 500 before the start of the transactions.

T_1	T_2
	sum = 0;
	avg = 0;
	Read(C);
	sum := sum + C;
	avg = 0;
Read(A);	
A := A – 50;	
Write(A);	
	Read(A);
	sum := sum + A;
	Read(B);
	sum := sum + B;
	avg := sum/3;
Read(B);	
B := B + 50;	
Write(B);	

Figure 12.4: Incorrect Summary Problem

Transaction T_2 reads the value of account A after A is updated and reads B before B is updated. If all the instructions in T_1 are executed before T_2 starts, then A will be 450, B will be 550 and average value will be 500.

If all the instructions in T_1 are executed after T_1 finishes, then A will be 450, B will be 550 and average value will be 500. But, due to this interleaved execution, the final value of A is 450, B is 550, and average is 483.33 which is wrong. Hence, the aggregate operation is end up with an inconsistent result. This problem is called as Inconsistent analysis *or incorrect summary problem*.

12.3.4 States of a Transaction

Figure 12.5 shows the different states of a transaction. When a transaction starts executing it immediately goes into an ***active*** state. It is the initial state of the transaction where it can issue READ and WRITE operations. When the transaction executes its final statement it moves to the **partially committed** state. If a transaction executes all its operations successfully and a transaction commit point is reached, it moves to **committed** state where all its effects of transaction operations are permanently established on the database system. If the transaction can no longer proceed further with normal execution then it is moved to the failed state.

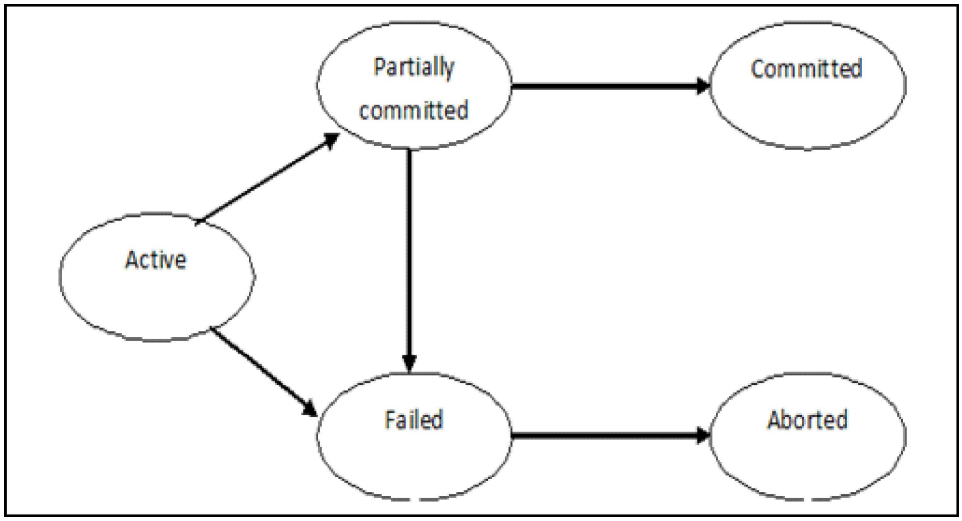


Figure 12.5: Sates of a transaction

However, if any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called **aborted**. In this stage system has two options:

- *Restart the transaction*: A restarted transaction is considered to be new transaction which may recover from possible failure.
- *Kill the transaction*: A transaction can be killed to recover from failure.

12.3.5 Desirable Properties of Transactions

To ensure the integrity of data during a transaction, the database system maintains the following properties. These properties are widely known as ACID properties:

- **Atomicity:** A transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. For example, let us consider a transaction where an account A has a balance of 200¹ and B has 300¹. Account A is transferring 100¹ to Account B. This is a transaction that has two operations: first, debiting 100¹ from A's balance and crediting 100¹ to B's balance. Let's say first operation passed successfully while second failed, in this case A's balance would be 100¹ while B would be having 300¹ instead of 400¹. This makes the account states inconsistent. Either the transaction should fail without executing any of the operation or it should process both the operations. The Atomicity property ensures that.
- **Consistency:** The transaction should be consistency preserving i.e. it should not have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

- **Isolation:** The property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will interfered by any other transaction executing concurrently.
- **Durability:** Once a transaction completes successfully (i.e. committed), the changes it has made into the database should be persistent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.



CHECK YOUR PROGRESS

Q.1: Choose the appropriate option:

i) Collections of operations that form a single logical unit of work are called as:

a) Views b) Networks
c) Units d) Transactions

ii) The "all-or-none" property is commonly referred to as:

a) Isolation b) Durability
c) Atomicity d) Consistency

iii) Execution of translation in isolation preserves the ..... of a database:

a) Atomicity b) Consistency
c) Durability d) All of the above

iv) Which of the following is not a transaction state?

a) Active b) Partially committed
c) Failed d) Compensated

12.3.6 The System Log and Commit Point of a Transaction

The transaction processing system maintains a **log** file i.e. **system log** to recover from failures that affect the transaction processing. It keeps track of all transaction operations that affect the values of database items which may be needed to permit recovery

from failures. The log is kept in the disk, so it is not affected by any type of failure except disk or catastrophic failure. It is periodically backed up to archival storage to guard against such catastrophic failures. The entries in the log file are known as **log records**. Following are the list of log records with its actions in a system log file where T refers to a unique **transaction id** which is generated automatically by the system:

- [**begin_transaction, T**]: Indicates that transaction *T* has started.
- [**write, T,Q, old_value, new_value**]: Indicates that transaction *T* has changed the value of the database item *Q* from *old_value* to *new_value*.
- [**read, T,Q**]: Indicates that transaction *T* has read the value of the database item *Q*.
- [**commit, T**]: Indicates that transaction *T* has completed successfully and confirms that its effect can be recorded permanently to the database.
- [**abort, T**]: Indicates that transaction *T* has been aborted.

Since the system logs maintain each READ and WRITE operations so, it is possible to undo or redo of operations whenever necessary for system recovery.

A transaction T reaches its **commit point** when all its operations have been executed successfully and the effect of all its operations on the database items have been recorded in the log. Beyond the commit point, the transaction is said to be **committed** and its effect is saved permanently into the database. At this point, the transaction writes a commit record **[commit, T]** into the system log file. If the transaction T, fails due to system failure, the recovery process search back in the log file for all transactions T that have written a **[begin_transaction, T]** record into the log but have not written **[commit, T]** record yet. These transactions may have to be rolled back to undo their effect on the database during the recovery process.

12.4 SCHEDULES OF TRANSACTIONS

Transactions can execute concurrently in interleaved fashion. The order of execution of operations from the various transactions is known as schedule. A **schedule** *S* of *n* transactions $T_1, T_2, ..., ..., ..., T_n$ is an ordering of the operations subject to the constraint that, for each transaction T_i that participates in *S*, the operations of T_i in *S* must appear in the same order in which they occur in T_i. However, operations from other transactions T_j can be interleaved with the operations of T_i in S. For the purpose of recovery and concurrency control, we are only interested on *Read*, *Write*, *Commit* and *Abort* operations of the transactions. For describing a schedule we can use a shorthand notations *r*, *w, c* and *a* for the operations *Read*, *Write*, *Commit* and *Abort* respectively, and can append the transaction id (i.e. transaction number) as subscript. So, the Schedule of fig. 12.2, which we shall call S_1 can be written as follows in the notation:

$$S_1: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Two operations in a schedule are said to be in ***conflict*** (conflicting pair) if they satisfy all of the following three conditions:

1) The operations belong to different transactions.
2) At least one of the operations is a *write (X*) operation.
3) The actions access the same item *X* (*read(X)* or *write(X)*).

So, the following set of actions is conflicting:

- $r_1(X); w_2(X); w_3(X);$ (3 conflicting pairs)

While the following sets of operations are not conflicting:

- $r_1(X); r_2(X); r_3(X)$ and $r_1(X); w_2(Y); r_3(X);$

A schedule *S* of *n* transactions $T_1, T_2, ..., ..., ..., T_n$ is said to be in ***complete schedule*** if they satisfy following conditions:

1) The operations in S are exactly those operations in $T_1, T_2, ..., ..., ..., T_n$ including a commit or abort operation as the last operation for each transaction in the schedule.
2) For any pair of operations from the same transaction T_i, their order of appearances in S is the same as their order of appearances in T_i.
3) For any two conflicting operations, one of the two must occur before the other in the schedule S.

12.4.1 Recoverability of Schedules

For some schedule, it is easy to recover from failure and some are not. A recoverable schedule is one where a committed transaction T should never be rolled back.

Recoverable Schedules: A **recoverable schedule** is one where, for each pair of Transaction T_i and T_j such that if T_j reads data item previously written by Ti, then the commit operation of T_i appears before the commit operation T_j.

For example let us consider the following schedule S as shown in figure 12.6. of two transactions T_1 and T_2 and let X = 100 initially. In this schedule, the transaction T_1 reads X = 100 and updates X as 110. The transaction T_2 reads the item X as 110 and updates as X = 105 and it is committed. Now, let T_2 fails due to some failure after T_2 committed. T_1 fails, so it is rolled back and set X to its original value as X = 100.

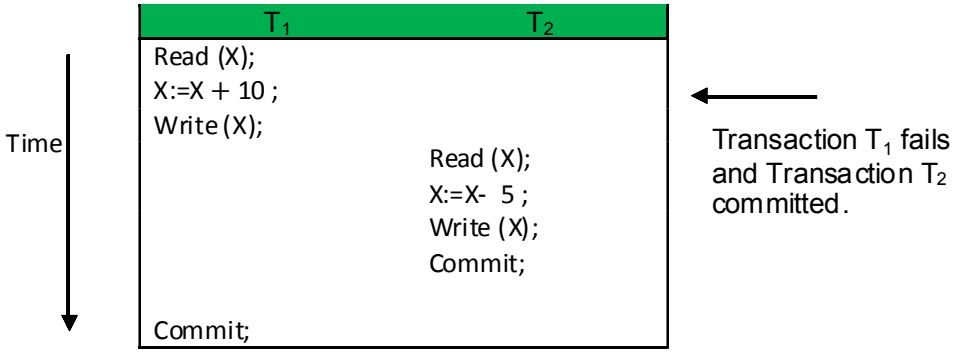


Figure 12.6: Non-Recoverable Schedule

Since, T_2 uses the value X updated by T_1, so it should also be rolled back which is not possible because a committed transaction never be rolled back. This schedule is therefore called as non-recoverable.

But, if commit operations of T_1 appears before commit operation of T_2 as shown in figure 12.7. Now, if transaction T_1 fails and rolled backed, T_2 has never to be rolled back due to T_1. It will always use the committed value of X and hence the schedule now becomes as recoverable.

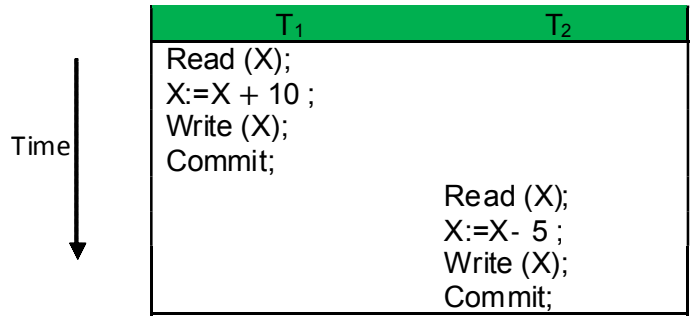


Figure 12.7: Recoverable Schedule

Cascadless Schedules: A cascading rollback occurs in database systems when a transaction T_1 causes a failure and a rollback must be performed. Other transactions dependent on T_1's actions must also be rollbacked due to T_1's failure, thus causing a cascading effect. This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called **Cascading rollback**. Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading rollbacks cannot occurs, Such schedules are called **Cascadeless Schedules.**

Formally, a *cascadeless schedule* is one where for each pair of transaction T_i and T_j such that T_j reads data item, previously written by T_i, the commit operation of T_i must appears before the read operation of T_j. Every Cascadeless schedule is also recoverable schedule.

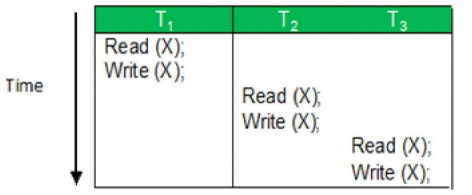


Figure 12.8: Cascading Rollback

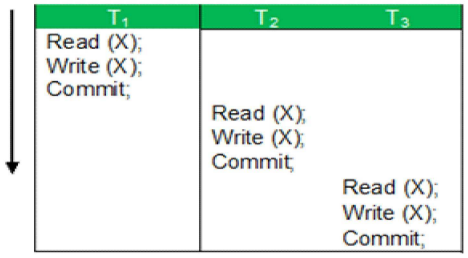


Figure 12.9: Cascading Schedule

Let us, take the example of a schedule S as shown in fig 12.8. Where, Transaction T_1 writes X that is read by Transaction T_2. Transaction T_2 writes X that is read by Transaction T_3. Suppose at this point T_1 fails. T_1 must be rolled back, since T_2 is dependent on

T_1, T_2 must be rolled back and since T_3 is dependent on T_2, T_3 must be rolled back. If we arrange the above transactions as a schedule as shown in Figure 12.9. it will avoid the cascading abort and such a schedule is known as **cascadeless** schedule.

Strict Schedule: A Strict schedule is one where transactions can *neither read nor write* an item X until the last transaction that wrote X has committed (or aborted). Strict schedule simplify the recovery process.

Following schedule as shown in figure 12.10 is a strict schedule.

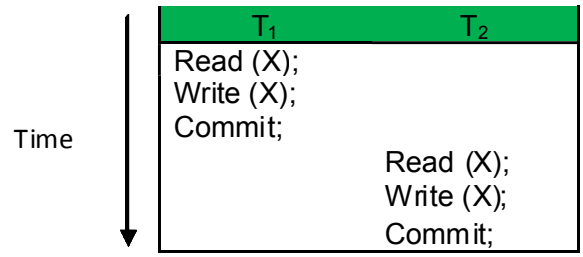


Figure 12.10: Strict Schedule

12.4.2 Serializability of Schedules

Serializability is the classical concurrency scheme. Serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedule. A schedule is called "correct" if we can find a serial schedule that is "equivalent" to it.

Serial Schedules: A schedule is called as a **serial** schedule when the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. Formally, a schedule S is serial if, for every transaction T_i participating in the schedule, all the operations of T_i are executed consecutively in the schedule; otherwise, the schedule is called **nonserial** schedule. In serial schedule only one transaction at a time is active. Figure 12.11(a) and 12.11(b) shows example of serial schedules S_1 and S_2 of transaction T_1 and T_2.

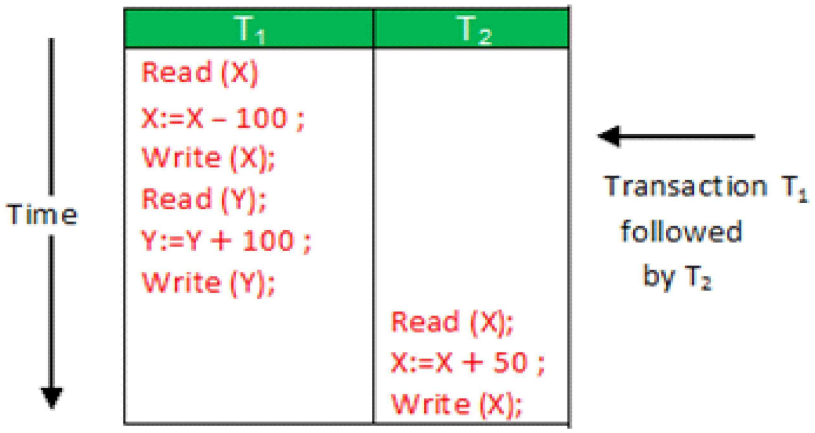


Figure 12.11(a): Serial Schedule S_1: followed by T_2

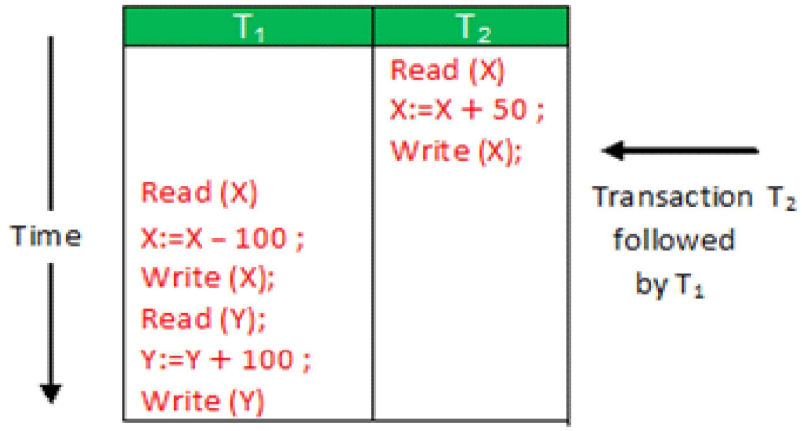


Figure 12.11(b): Serial Schedule S_2: followed by T_1

Figure 12.11(c) and 12.11(d) shows example of non-serial schedules S_3 and S_4 of transaction T_1 and T_2 with interleaving of operations of T_1 and T_2.

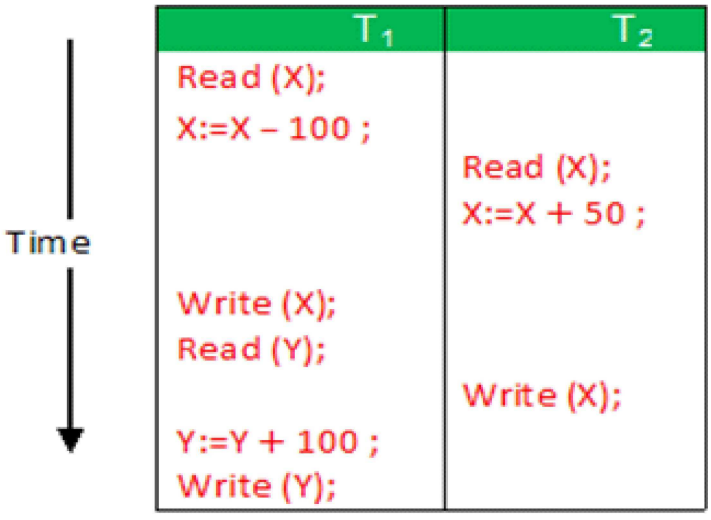


Figure 12.11(c): Non-Serial Schedule S_3 with interleaving of T_1 and T_2

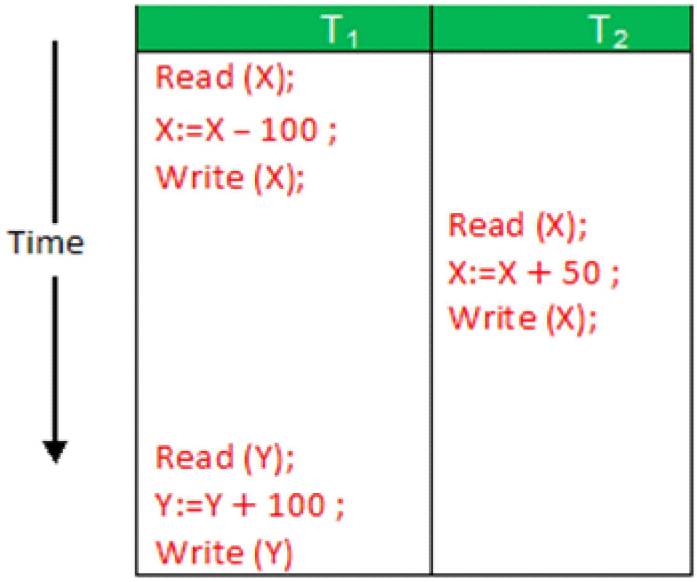


Figure 12.11(d): Non-Serial Schedule S_4 with interleaving of T_1 and T_2

Serializable Schedule: A serial schedule is inefficient because no interleaving is allowed. This leads to low CPU utilization, and slowing down processes in general. A serializable schedule gives the benefits of concurrent execution without giving up correctness. A schedule S of n transactions $T_1, T_2, ..., ..., ..., T_n$ is **serializable** if it is **equivalent** to some serial schedule of the same n transactions $T_1, T_2, ..., ..., ..., T_n$. There are n! possible serial schedules of n transactions and many more possible nonserial schedules.

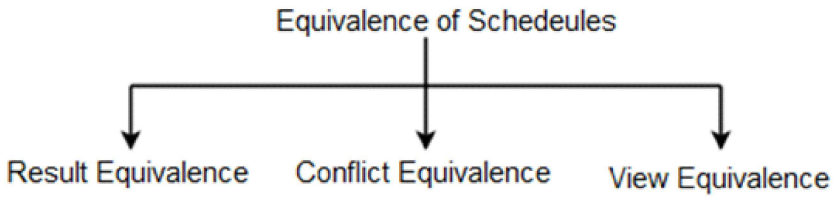


Figure 12.12: Equivalence of Schedules

A non-serial schedule S is equivalence schedule to a serial schedule if both are correct.

Schedules may have three different kinds of equivalence schedules among them which are: **Result Equivalence**, **Conflict Equivalence** and **View Equivalence**.

- **Result Equivalence:** If any two schedules generate the same result after their execution, then such schedules are called as **result equivalent schedules**. This equivalence relation is

considered of least significance because some schedules might produce the same results for some set of values and they might produce the different results for some other set of values.

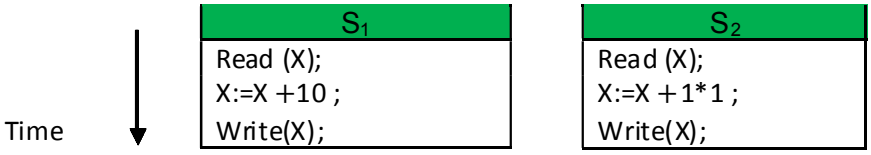


Figure 12.13: Non Result Equivalence

Two schedules S1 and S_2 are result equivalent for initial value but are not result equivalent in general.

- **Conflict Equivalence:** Two schedules are conflict equivalent if the order of any two conflicting operations is the same in both schedules. This is the more commonly used definition of equivalence. A schedule is conflict serializable if it is equivalent to some serial schedule. We can reorder the nonconflicting operations if necessary. A schedule is called **conflict serializable** if it can be transformed into a serial schedule by swapping non-conflicting operations. The definition of conflict operations are discussed in section 12.4.

  Consider the schedule S_4 of figure 12.11(d) which is equivalent to the serial schedule S_1 of figure 12.11(a). In both the schedule read(X) of T_2 reads the value of X written by T_1, while other read operations read the values from the initial database state. Furthermore, T_1 is the last transaction to write Y, and T_2 is the last transaction to write X in both the schedules. Since, schedule S_1 is a serial schedule and schedule S_4 is equivalent to S_1. So, S_4 is a serial schedule. Now, notice that the operations $R_1(X)$ and $W_1(X)$ of schedule S_4 do not conflict with the operations $R_1(X)$ and $W_1(X)$ of schedule S_1, since they access different data items. Hence, we can move $R_2(X)$, $W_2(X)$ before $R_2(X)$, $W_2(X)$, leading to the equivalent serial schedule T_1, T_2. Hence, Schedule S_4 is conflict serializable schedule to schedule S_1.

- **View Equivalence:** Two schedules are said to be view equivalence, if the transaction in one schedule is same as the

one in other. That means, both the transactions in two schedules perform same tasks. Formally, two schedules Si and Sj are said to be view equivalent if the following three conditions hold:

- The same set of operations participates in S_i and S_j, and S_i and S_j include the same operations of those transactions.
- For any operation $R_i(Q)$ of T_i in S_i, if the value of Q read by the operation has been written by the operation $W_j(Q)$ of T_j (or if it is the original value of Q before the schedule started), the same condition must hold for the value of Q read by operation $R_i(Q)$ of T_i in S_j.
- If the operation $W_k(P)$ of T_k is the last operation to write item P in $_{Si}$, then $W_k(P)$ of T_k must also be the last operation to write item P in S_j.

A schedule S is said to view serializable if it is view equivalent to a serial schedule.



CHECK YOUR PROGRESS

Q.2: Choose the appropriate option:

i) If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, then S and S' are:

a) Non conflict equivalent b) Equal

c) Conflict equivalent d) Isolation equivalent

ii) A schedule is ..... if it is conflict equivalent to a serial schedule.

a) Conflict serializable b) Conflicting

c) Non serializable d) None of the mentioned

iii) The phenomenon in which one failure leads to a series of transaction rollbacks is called as .....

a) Cascadeless rollback b) Cascading rollback

c) Cascade cause d) None of the mentioned

iv) If a schedule is equivalent to a serial schedule, it is called as a .....

a) Serializable schedule b) Equivalent schedule

c) Committed schedule d) Conflicting Schedule



12.5 LET US SUM UP

- A transaction is a set of operations that must be processed as one unit of work.
- Transaction always follows **A**tomicity, **C**onsistency, **I**solation and **D**urability properties
- A transaction may be any one of the five states such as *Active*, *Partially Committed*, *Committed*, *Failed* and *Abort*.
- Concurrency control is needed to avoid *Dirty Read*, *Lost Update* and *Incorrect Summary* problems
- Transactions can be executed concurrently in interleaved fashion and the order of execution of operations from the various transactions is known as schedule.
- Scheduling of transaction is important for concurrency control and can be done in different ways.
- Schedules of transactions may be serial or non-serial.
- There may various ways to find out equivalence of schedules and serialization of schedule is equivalent to some serial schedule.
- There are three types of equivalence of transaction schedules: *Result* equivalence, *Conflict* equivalent and *View* equivalent.
- There are two types of serialization of schedules: Conflict serializability and View serialzability based on Conflict equivalent and view equivalent respectively.



12.6 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.

2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



12.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (d), ii) (c), iii) (b), iv) (d)

Ans. to Q. No. 2: i) (c), ii) (a), iii) (b), iv) (a)



12.8 MODEL QUESTIONS

Q.1: With example, explain single user and multi-user system

Q.2: What do you mean by a transaction? Explain the desirable properties of a transaction.

Q.3: What do you mean by the commit point of a transaction?

Q.4: Why concurrency control is needed in transaction processing? Explain briefly *dirty read, incorrect summary* problem.

Q.5: Explain *Lost update* problem with example.

Q.6: What is system log? Explain the operations performed in the system log.

Q.7: Draw a state diagram and explain the typical states that a transaction goes through the execution.

Q.8: What is a schedule? Define the concepts of *recoverable*, *cascadeless* and *strict schedules*, and compare them in terms of recoverability.

Q.9: What is serializability of transactions? Explain different types of serializability.

Q.10: What are the conditions for conflict equivalent schedules? Explain conflict serializability.

Q.11: Consider the following schedule S_1 of two transactions T_1 and T_2. Find out whether the schedule S_1 is conflict serializable or not.

$S_1 = r_1(Q); r_2(Q); w_1(Q); w_2(Q);$

*** ***** ***

UNIT 13: CONCURRENCY CONTROL

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Two Phase Locking Techniques for Concurrency Control

13.4 Types of Two Phase Locking

- 13.4.1 Conservative 2PL
- 13.4.2 Strict Two Phase Locking
- 13.4.3 Distributed Two Phase Locking

13.5 Concurrency Control based on Timestamp Ordering

- 13.5.1 Basic Timestamp Ordering
- 13.5.2 Strict Timestamp Ordering

13.6 Multi-Version Concurrency Control (MVCC)

- 13.6.1 The Schedulers for MVCC based Timestamp Ordering
- 13.6.2 Multiversion Two-Phase Locking

13.7 Validation (Optimistic) Concurrency Control Techniques

13.8 Granularity of Data Item

13.9 Database Recovery Concept

- 13.9.1 Category of Recovery Algorithm

13.10 Let Us Sum Up

13.11 Further Readings

13.12 Answer to Check Your Progress

13.13 Model Questions

13.1 LEARNING OBJECTIVES

After going through this unit, you will able to:

- define Concurrency
- learn about the differrent Concurrency control techniques
- describe the idea of Controlling Concurrency based on Timestamp
- descibe the types of timestamp ordering

- learn about multi-version Concurrency control and Optimistic Concurrency control techniques
- describe the concept of granularity.

13.2 INTRODUCTION

In the previous unit, we learnt about transaction and serializability of transaction. This unit discusses the different concurrency control techniques and locking and how locking helps in transaction isolation. Most of the techniques discussed ensure the serializabality by using different protocols. The unit discusses the techniques like two phase locking, timestamp ordering, optimistic protocols etc. The effect of granularity in currency control is also discussed at the end of this unit.

13.3 TWO PHASE LOCKING TECHNIQUES FOR CONCURRENCY CONTROL

In case of shared data, locking is a mechanism commonly used to solve the problem of synchronized access. Conceptually, it is a variable associated with a data item. It is important here to note that each data item has a lock associated with it. Before a transaction T1, access a data item, the scheduler first examines the associated lock with the data item. If no transaction holds the lock on the data item, then the scheduler obtains the lock on behalf of T1. If another transaction T2, hold the lock, then T1, has to wait until T2 gives up the lock. That is, the scheduler will not give T1, the lock until T2, releases it. The scheduler thereby ensures that only one transaction can hold the lock at a time, so only one transaction can access the data item at a time. Locking can be used by a scheduler to ensure serializability of transaction. To present such a locking protocol, we need some notation.

Transactions access data items either for reading or for writing them. We therefore associate two types of locks with data items: read locks and write locks. Let us assume that rl[x] denote a read lock on data item x and wl[x] denote a write Iock on x. rl[x] (or wl[x]) to indicate that transaction T, has obtained a read (or write) lock on x. We use $rl_i[x]$ (or $wl_i[x]$) to denote the operation by which T_i releases its read (or write) lock on x. In this case, we

say T_i unlocks x. It is the job of a two Phase Locking (2PL) scheduler to manage the locks by controlling when transactions obtain and release their locks. A transaction is said to follow the Two Phase Locking protocol, if all locking operations (either read or write lock) first perform the unlock operation in any transaction. The rules according to which a basic 2PL scheduler manages and uses its locks:

1) When it receives an operation $p_i[x]$ from the Transaction Manager, the scheduler tests if $pl_i[x]$ conflicts with some $ql_i[x]$ that is already set. If so, it delays $p_i[x]$, forcing T_i to wait until it can set the lock it needs. If not, then the scheduler sets $pl_i[x]$, and then sends $p_i[x]$ to the Database Manager.
2) Once the scheduler has set a lock for T_i, say $pl_i[x]$, it may not release that lock at least until after the Database Manager acknowledges that it has processed the lock's corresponding operation, pi[x].
3) Once the scheduler has released a lock for a transaction, it may not subsequently obtain any more locks for that transaction on any data item.

From the above rule it has been noticed rule (1), prevents two transactions from concurrently accessing a data item in conflicting modes. Thus, conflicting operations are scheduled in the same order in which the corresponding locks are obtained.

The rule (2) supports rule (1) by ensuring that the Database Manager processes operations on a data item in the order that the scheduler submits them. For example, suppose T_i obtains rl[x], which it releases before the Database Manager has confirmed that $r_i[x]$ has been processed. Then it is possible for T_j to obtain a conflicting lock on x, $wl_j[x]$, and send $w_j[x]$ to the Database Manager. Although the scheduler has sent the Database Manager $r_i[x]$ before $w_j[x]$. Without rule (2) there is no guarantee that the Database Manager will receive and process the operations in that order.

Rule (3), also called the two phase rule, is the base for the name two phase locking. Each transaction may be divided into two phases: a growing phase or expanding phase (first phase) during which it obtains locks, and a shrinking phase (second phase) during which it releases locks.

13.4 TYPES OF TWO PHASE LOCKING

There are basically two types of Two Phase Locking. They are:

- Conservative Two Phase Locking
- Strict Two Phase Locking

13.4.1 Conservative Two Phase Locking

It is possible to construct a Two Phase Locking scheduler that never aborts transactions. This technique is known as **Conservative Two Phase Locking** or **Static 2X**. As Two Phase Locking most of the time causes abortions because of deadlocks. Conservative Two Phase Locking avoids deadlocks by requiring each transaction to obtain all of its locks before any of its operations are submitted to the Database Manager. This is done by having each transaction to declare its readset and writeset in advance. Specifically, each transaction T_i first tells the scheduler all the data items it will want to Read or Write, for example as part of its Start operation. The scheduler tries to set all of the locks needed by T_i. It can do this by ensuring that none of these locks conflicts with a lock held by any other transaction. If the scheduler succeeds in setting all of T_i's locks, then it submits T_i's operations to the Database Manager as soon as it receives them. After the Database Manager acknowledges the processing of T_i's last database operation, the scheduler may release all of T_i's locks. If, on the other hand, any of the locks requested in T_i's Start conflicts with locks presently held by other transactions, then the scheduler does not grant any of T_i's locks. Instead, it inserts T_i along with its lock requests into a waiting queue. Every time the scheduler releases the locks of a completed transaction, it examines the waiting queue to see if it can grant all of the lock requests of any waiting transactions. If so, it then sets all of the locks for each such transaction and continues processing as just described. In Conservative Two Phase Locking, if a transaction T_i is waiting for a lock held by T_j, then T_i is holding no locks. Therefore,

no other transaction T_k can be waiting for T_i. Since deadlock is the only reason that a Two Phase Locking scheduler ever rejects an operation and thereby causes the corresponding transaction to abort, Conservative Two Phase Locking never aborts a transaction. By delaying operations sooner than it has to, namely, when the transaction begins executing, the scheduler avoids abortions that might otherwise be needed for concurrency contro1 reasons.

13.4.2 Strict Two Phase Locking

Almost all implementations of Two Phase Locking use a variant called **Strict Two Phase Locking**. This differs from the Basic 2PL scheduler described in the previous section that it requires the scheduler to release all of a transaction's locks together, when the transaction terminates. More specifically, T_i's locks are released after the Database Manager acknowledges the processing of c_i or a_j, depending on whether T_i commits or aborts respectively. The reasons for adopting this policy is Firstly, consider when a Two Phase Locking scheduler can release some $ol_i[x]$. To do so the scheduler must know that: (1) T_i has set all of the locks it will ever need, and (2) T_i will not subsequently issue operations that refer to X. One point in time at which the scheduler can be sure of (1) and (2) is when T_i terminates, that is, when the scheduler receives the c_i or a_i operation. In fact, in the absence of any information from the Transaction Manager aside from the operations submitted, this is the earliest time at which the scheduler can be assured that (1) and (2) hold. A second reason for the scheduler to keep a transaction's locks until it ends, and specifically until after the Database Manager processes the transaction's Commit or Abort, is to guarantee a strict execution.

13.4.3 Distributed Two Phase Locking

Two phase locking can also be realized in a distributed Database Management System. A distributed Database Management System consists of a collection of communicating nodes, each of

which is a centralized Database Management System. Each data item is stored at exactly one node. The scheduler manages the data items stored at its site. This means that the scheduler is responsible for controlling access to the items. A transaction submits its operations to a Transaction Manager. The Transaction Manager then delivers each Read(x) or Write(x) operation of that transaction to the scheduler that manages x. When a scheduler decides to process the Read(x) or Write(x), it sends the operation to its local Database Manager, which can access x and return its value (for a Read) or update it (for a Write). The Commit or Abort operation is sent to all nodes where the transaction accessed data items. The schedulers at all nodes, taken together, constitute a distributed scheduler. The task of the distributed scheduler is to process the operations submitted by the Transaction Managers in a globally serializable and recoverable manner. It is possible to build a distributed scheduler based on Two Phase Locking. Each scheduler maintains the locks for the data items stored at its node and manages them according to the Two Phase Locking rules. In Two Phase Locking, a Read(x) or Write(x) is processed when the appropriate lock on x can be obtained, which only depends on what other locks on x are presently owned. Therefore, each local Two Phase Locking scheduler has all the information it needs to decide when to process an operation, without communicating with the other sites. It is problematic to decide when to release a lock. To enforce the two phase rule, a scheduler cannot release a transaction T_i's lock until it knows that T, will not submit any more operations to it or any other scheduler. Otherwise, one scheduler might release T_i's lock and sometime later another scheduler might set a lock for T_i, thereby violating the two phase rule.

It would appear that enforcing the two phase rule requires communication among the schedulers at different nodes. However, if schedulers use Strict Two Phase Locking, then they can avoid such communication. Because, the Transaction Manager that manages transaction T, sends T_i's Commit to all nodes where T_i accessed

data items. By the time the Transaction Manager decides to send T_i's Commit to all those sites, it must have received acknowledgments to all of T_j's operations. Therefore, T_i has surely obtained all the locks it will ever need. Thus, if a scheduler releases T_i's locks after it has processed T_j's Commit (as it must under Strict Two Phase Locking), it knows that no scheduler will subsequently set any locks for T_i. To prove that the distributed Two Phase Locking scheduler is correct we simply ensure that any history H it could have produced satisfies the properties of Two Phase Locking histories.



CHECK YOUR PROGRESS

Q.1: Fill up the gapes:

a) A lock is a ..... associated with a data item.

b) Changing the state of lock from one type to another is known as .....

c) In ..... phase of locking, new locks on item can be granted.

d) In ..... phase, existing locks are released.

e) In conservative two phase locking technique, it never ..... transaction.

13.5 CONCURRENCY CONTROL BASED ON TIMESTAMP ORDERING

In timestamp ordering, the Transaction Manager assigns a unique timestamp, $ts(T_i)$, to each transaction T_i. It generates timestamps using some of the efficient techniques, in the context of timestamp-based deadlock prevention. The Transaction Manager attaches a transaction's timestamp to each operation issued by the transaction. It will therefore be convenient to speak of the timestamp of an operation $o_i[x]$, which is simply the timestamp of the transaction that issued the operation. A **Timestamp Ordering (TO)** scheduler orders conflicting operations according to their timestamps. More precisely, it applies the following rule, called the **TO rule**.

TO Rule: If $p_i[x]$ and $q_i[x]$ are conflicting operations, then the Database Manager processes $p_i[x]$ before $q_i[x]$ if and only if $ts(T_i) < ts(T_i)$. By enforcing the TO rule, it can be ensured that every pair of conflicting operations is executed in timestamp order. Thus, a TO execution has the same effect as a serial execution in which the transactions appear in timestamp order.

13.5.1 Basic Timestamp Ordering

The Basic TO is a simple and aggressive implementation of the TO rule. It accepts operations from the Transaction Manager and immediately outputs them to the Database Manager in first-come-first-served order. To ensure that this order does not violate the TO rule, the scheduler rejects operations that it receives too late. An operation $p_i[x]$ is too late if it arrives after the scheduler has already output some conflicting operation $q_i[x]$ with $ts(T_i) > ts(T_j)$. If $p_i[x]$ is too late, then it cannot be scheduled without violating the TO rule. Since the scheduler has already output $q_j[x]$, it can only solve the problem by rejecting $p_i[x]$.

If $e_i[x]$ is rejected, then T_i must abort. When T_i is submitted again, it must be assigned a timestamp sufficiently large so that its operations are less likely to be rejected during its second execution.

To determine if an operation has arrived too late, the Basic TO scheduler maintains for every data item x the maximum timestamps of Reads and Writes on x that it has sent to the Database Manager, denoted by max-r-scheduled[x] and max-w-scheduled[x] respectively. When the scheduler receives $e_i[x]$, it compares $ts(T_i)$ to max-q-scheduled[x] for all operation types q that conflict with p. If $ts(T_i) <$ max-q-scheduled[x], then the scheduler rejects $e_i[x]$, since it has already scheduled a conflicting operation with a larger timestamp. Otherwise, it schedules $e_i[x]$ and, if $ts(T_i) >$ max-p-scheduled[x], it updates max-p-scheduled[x] to $ts(T_i)$.

The scheduler must communicate to the Database Manager to guarantee that operations are processed by the Database

Manager in the order that the scheduler sent them. Even if the scheduler decides that $e_i[x]$ can be scheduled, it must not send it to the Database Manager until every conflicting $q_j[x]$ that it previously sent has been acknowledged by the Database Manager.

For proper maintenance of the communication with the Database Manager, the Basic TO scheduler also maintains, for each data item x, the number of Reads and Writes that have been sent to the Database Manager, but not yet acknowledged. These are denoted as r-in-transit[x] and w-in-transit[x] respectively. For each data item x the scheduler also maintains a queue, queue[x], of operations that can be scheduled in so far as the TO rule is concerned, but are waiting for acknowledgments from the Database Manager to previously sent conflicting operations.

13.5.2 Strict Timestamp Ordering

An improved version of Basic TO is the Strict TO, which is both Strict ensuring easy recoverability and serializable. Although the TO rule enforces serializability, it does not necessarily ensure recoverability. If w-in-transit[x] denotes the number of w[x] operations that the scheduler has sent to the Database Manager but that the Database Manager has not yet acknowledged. Since two conflicting operations cannot be “in transit” at any time and Writes on the same data item conflict, w-in-transit[x] at any time is either 0 or 1. The Strict TO scheduler is similar to Basic TO in every respect, except that it does not set w-in-transit[x] to 0 when it receives the Database Mamnager’s acknowledgment of a w[x]. Instead it waits until it has received acknowledgment of a_i or c_i. It then sets w-in-transit[x] to zero for every x for which it had sent $w_i[x]$ to the Database Manager. This delays all $r_j[x]$ and $w_j[x]$ operations with $ts(T_j) > ts(T_i)$ until T_i has committed or aborted. This means that the execution output by the scheduler to the Database Manager is strict.



CHECK YOUR PROGRESS

Q.2: State True or False:

a) Timestamp is an unique identifier used to identify a transaction when it is aborted.

b) Arranging of transaction based on timestamp value is known as Timestamp ordering.

13.6 MULTI-VERSION CONCURRENCY CONTROL (MVCC)

In a multiversion concurrency control algorithm, each Write on a data item x produces a new copy or version of x, thereby keeping the original copy intact. The Database Manager that manages x keeps a list of copies or versions of x, which is the history of values that the Database Manager has assigned to x. For each Read(x), the scheduler not only decides when to send the Read to the Database Manager, but it also tells the Database Manager which one of the copy or version of x to read.

The main benefit of multiple versions for controlling concurrency is to help the scheduler avoid rejecting operations that arrive too late. For example, the scheduler normally rejects a Read because the value it was supposed to read has already been overwritten. With multiversions, such old values are never overwritten and are therefore always available for Reads. The Database Manager makes the different versions explicitly available to the scheduler. The disadvantage of MVCC is the utilization of storage space in turn wasting valuable memory space and for better processing, it is also important to do the archiving thereby incurring processing overhead.

It is important to note here that when a transaction is aborted, the scheduler does not create a new version. The new versions are only created for the transaction which is either active or committed. Thus, when the scheduler decides to assign a particular version of x to Read(x), the value returned is not one produced by an aborted transaction.

13.6.1 The schedulers for MVCC based Timestamp Ordering

According the TO algorithm, each transaction has a unique timestamp, denoted ts(T_i). Each operation carries the timestamp of its corresponding transaction. Each version is labeled by the timestamp of the transaction that wrote it. A Multiversion TO (M VTO) scheduler processes operations on first-come-first-served basis. The scheduler processes $r_i[x]$ by first translating it into $r_i[x_k]$, where x_k is the version of x with the largest timestamp less than or equal to ts(T_i), and then sending $r_j[x_k]$ to the Database Manager. It processes $w_i[X]$ by considering two cases.

If it has already processed a Read $r_j[x_k]$ such that ts(T_k) < ts(T_i) < ts(T_j), then it rejects $w_i[x]$. Otherwise, it translates $w_i[x]$ into $w_j[x_j]$ and sends it to the Database Manager. Finally, to ensure recoverability, the scheduler must delay the processing of c_i until it has processed C_j for all transactions T_j that wrote versions read by T_i.

13.6.2 Multiversion Two-Phase Locking

According to this concept there are three locking modes read, write certify instead of two, namely read and write. So, the state of LOCK(X) for an item X can be read-lock, write-lock or certify. According to the locking concept, write_lock is considered as an exclusive lock, i.e. when a transaction holds a write_lock on an item, that item cannot be locked by any other transaction.

The idea behind multiversion Two Phase Locking 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 having two versions for each item x. One version must always have been written by some committed transaction, while the second version is created when a transaction T acquires a write lock on the item. So, other transaction can continue accessing the item from the committed version of x. Any transaction T can write the value of x' when required,

without affecting the value of the committed version of x. 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. To obtain a certify lock, a transaction may have to delay its commit, until all its write locked items are released by any reading transactions. Once certify lock is obtained, the committed version of the data is updated with the new value and the certify lock is released.

Thus, in case of multiversion Two Phase Locking, read can proceed concurrently with a single write operation, which is not permitted in case of standard Two Phase Locking.



CHECK YOUR PROGRESS

Q.3: Fill up the gapes:

a) In Multi Version Concurrency Control, old value of data items are kept, even after ..... of data item.

b) In Multi Version two phase locking, ... types of locking is present.

c) In Multi Version Concurrency Control, a ..... version is created, whenever there is an update.

13.7 VALIDATION (OPTIMISTIC) CONCURRENCY CONTROL TECHNIQUES

In all the concurrency control techniques that we have discussed so far, some kind of checking is done before a database operation can be executed. This kind of checking adversely affects the performance of the database by slowing down the transaction execution process. For example, in case of locking, a check is done to verify whether the item accessed by the transaction is locked or not. This gives extra overhead, thereby slowing down the transaction execution time.

In case of validation concurrency control technique, which is also known as Optimistic concurrency control technique, no checking is done while the transaction is executing. That is, a transaction is started without

locking the database item. For example, in case of update operation in a transaction, it is not directly applied to the database, until the transaction reaches the end. During the execution phase of a transaction, all updates are applied to the local copy of the database. when transaction reaches the end of the execution, it goes through a phase known as **validation phase,** which checks whether any of the transaction update violate serializability. The information needed by the validation phase is kept in the system.

If serializability is not broken or violated, the transaction is committed and the database is updated from the local copies, otherwise, the transaction is aborted and restarted later.

The different phases of validation check concurrency control protocol are:

1) **Read Phase:** A transaction read values of committed data items from the database. However, because transaction is committed, the updates or writes are applied only to the local copies of the data item kept in the transaction workspace.
2) **Validation Phase:** Checking is performed to ensure that update from a transaction will not violate the serializability.
3) **Write Phase:** Based on the status of the validation phase, if the validation phase is successful, the transaction updates are applied to the database and if it is unsuccessful, the updates are discarded and the transaction is restarted later.

In *optimistic concurrency control* technique, a transaction proceeds without any checking, that is, if proceeds with a minimum overhead, until it reaches the validation phase. If the interference between the transaction is less, they are validated successfully, and result is updated to the database. On the other hand, when the interference among the transaction is high, it results in unsuccessful validation, and the results are discarded and restarted later. The optimistic technique assumes that there is less interference among the transactions and it hardly needs any checking during the transaction execution.

In optimistic algorithm, during the validation phase of the Transaction T_i, the algorithm checks that T_i does not interfere with any committed

transactions or with any transaction which is currently running in their validation phase. The validation phase of ant transaction T_i, checks that for any other transaction T_j that is either committed or it is in its validation phase holds one of the following conditions:

1) T_j completes its write phase, before T_i starts its read phase.
2) T_i starts its write phase only after T_j completes its write phase and read_set of T_i has no item in common with the write_set of T_j.
3) The read_set and write_set of T_i have no item is common with the write_set of T_j and T_i completes its read phase before T_i completes its read phase.

The above three conditions checked serially, and stars with the condition 1. If it is false, it checks the condition 2, and if it is also false, then only it checks the condition 3. If any one of the above three conditions held, it means there is no interference and T_i is validated successfully. On the other hand, it none of the above three conditions holds, the validation phase of T_i fails and is aborted and restarted later.



CHECK YOUR PROGRESS

Q.4: State True or False:

a) Optimistic Concurrency Control is also known as validation 1 technique.

b) In Optimistic Concurrency Control technique, a check is done while a transaction is executing.

c) In read phase, a transaction read values of committed data items from the database.

d) Validation phase ensures that serializability is not maintained.

e) In Optimistic Concurrency Control, transaction execution proceeds with a minimum overhead until the validation phase is reached.

13.8 GRANULARITY OF DATA ITEM

All the concurrency control technique categorize a database in the form of a set of data items. The different data items that may be considered are as follows:

a) It may as small as a field value of a database record.
b) It may be a database record.
c) A disk block.
d) A whole file.
e) A whole database.

The granularity of database is very important as far as the performance of concurrency and recovery is concerned.

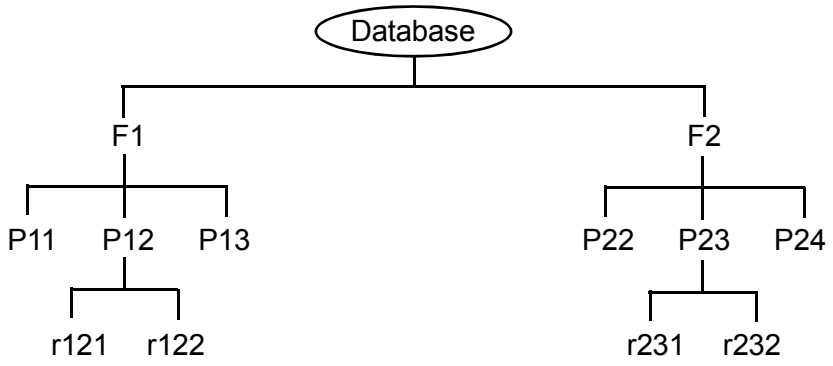
The size of the data item is called *data item granularity*. The granularity can be considered as– *fine granularity*, which refers to the smallest size of the data items and ***coarse granularity*** which refers to the largest size of the data items.

It is to be mentioned here that the degree of concurrency is dependent on the granularity of data item. When the size of the data item is large, it lowers the degree of concurrency, so the number of concurrent transaction is less. This is because, when the entire disk block is locked by a transaction, any other transaction trying to access items belonging to the same disk block is forced to wait. On the other hand, if the database size is small, for example if it is a single record only, then many transactions can proceed parallelly, because it would be locking a different data item record.

On the other hand, when the size of the data item is small, more number of data items are present in the database. Since, every item is associated with a lock, there will be more number of locks and the lock manager need to manage a large number of locks which may be active at the same time. Since, more number of lock and unlock operations need to be performed, It increases the overall overhead on the database. At the same time, the size of the lock table also increases.

The appropriate size of the data item is very difficult to decide on and it depends on the type and number of transactions executing at one

point of time. As the best size of the granularity depends on the transaction, so database system should support the different level of granularity for a set of transactions. A hierarchy of granularity is depicted in the figure below:



Suppose a transaction T1 wants to update some of the records under the file F1. So, it will lock all the items down the hierarchy starting from F1. Now, suppose another transaction T2 want to read a record, which is also belongs to the file F1. The transaction manager of the Database System will check the compatibility of the requested lock, with the locks that are already held. While verifying the compatibility, if at any point of time, a conflict occurs, the request for the read(shared) lock is denied and the transaction T2 is denied and must wait.

But when T2 request for the read (shared) lock comes first, T2's request is granted. At this time, it is very difficult to check all the need for a lock conflict, and the time consumption is also very high. So, it becomes very much in efficient and in tern defeat the purpose of having multiple granularity locks.

To overcome the above problem and to implement multiple granularity level locking, another type of lock known as ***intention lock*** is implemented. The objective of this type of lock is to indicate along the path from the root to the desired node, what type of lock it would required. There are three types of intention locks namely,

1) Intention-Shared (IS)– which indicates that a shared lock will be requested on some descendent nodes.
2) Intention-Exclusive (IX)– which indicates that a shared lock will be requested on some descendent nodes.

3) Shared-Intention-Exclusive (SIX)– it indicates that the current node is locked in shared lock but an exclusive lock will be requested on some descendent nodes.

Apart from the different types of intention lock some appropriate locking protocol also need to be implemented for better performance.



CHECK YOUR PROGRESS

Q.5: Fill up the gapes

a) Field value is the ..... granularity of data item.

b) Fine grain refers to the ..... data item size.

c) ..... grain refers to the large data item size.

d) Best data item size depends on the ..... type involved.

e) Intention lock is a type of lock applied to ..... granularity lock.

13.9 DATABASE RECOVERY CONCEPT

Recovery from transaction failure can be defined as the restoring of the database to the most recent consistent state just before the time of failure. To ensure this, the system must keep a record of all the changes that took place to the database due to the execution of various transactions. This information is maintained by the transaction log. In case of failure, with the help of this lock, the database can be recovered.

13.9.1 Category of Recovery Algorithm

There may be various possibilities due to which recovery becomes an important issue. In case of huge damage, which is mainly caused due to the catastrophic failure, the database can be recovered by restoring the database kept as a backup or from the archive and reconstruct a more current state by reapplying or redoing the operations of committed transaction from the backup log up to the

time of failure. On the other hand when the database is not physically damaged but has become inconsistent due to various inconsistencies by undoing some operations. There may be a need of redoing some of the operation in order to restore a consistent state of the database.

a) **Deferred Update and Immediate Update:** To recover from the non-catastrophic failure, there are two main techniques applied. They are:

   i) **Deferred Update:** According to deferred update, the database is not physically updated to the disk until a database reaches its commit point. When a transaction reaches a commit point, then only the transaction update is recorded. Before reaching the commit point, all transaction updates are recorded to the local database. In case transaction fails before reaching the commit point, it is not required to make any changes to that database, so no undo is needed. There may be a necessity to redo some of the operations from the transaction logs. Because their effect may not have been recorded in the database. So, deferred update is also sometime known as no-undo/redo algorithm.

   ii) **Immediate Update:** In this technique, the database may be update by some operations of the transaction before the transaction reaches its commit point. However, these operations are also recorded in the log on the disk by force writing before they are applied to the database, thereby making the recovery still possible. In case a transaction fails before reaching the commit point, the effect of its operation on the database must be undone i.e. the transaction must be rolled back. So, in case of immediate update, both undo and redo is needed. So, this algorithm is also known as **UNDO/REDO** algorithm. There may be another variation of the algorithm where all updates are recorded in the database before a transaction is committed requiring undo only, so it is known as undo/no-redo algorithm.

b) Caching of disk Blocks: The caching of disk block typically is an operating system function. But as it is one of the important parameter as far as the efficiency of recovery procedure is concerned, it is handled by the DBMS by calling low level operating system routines. Typically a collection of in-memory buffer called the DBMS cache, is kept under the control of the DBMS for the purpose of holding these buffers. Associated with each buffer in the cache is a **dirty bit**, which can be included in the directory entry to indicate whether or not the buffer has been modified. When a page is first read from the database disk into a cache buffer, the cache directory is updated with the new disk address, the dirty bit is set to 0. As soon as the buffer is modified the dirty bit for the corresponding directory entry is set to 1. When the buffer content are replaced from the cache, the content must first be written back to the corresponding disk page only if its dirty bit is set to 1.

Another bit known as pin-unpin bit also used to indicate whether a page in the cache can be written back to the disk or not. Here, pinned (bit value 1) indicates page cannot be written back.

As far as flashing of modified buffer back to the disk is concerned, two main strategies are employed. They are:

i) In-place updating

ii) Shadowing

In case of in-place updating, it writes the buffer to the same original disk location, thus overwriting the old value of any changed data item on disk. Hence, only a single copy of each database disk block is maintained.

In the case of shadowing on the other hand, it writes an updated buffer at a different disk location, so multiple versions of data items can be maintained. The old value of data item before updating is called the before image (BFIM) and the new value after update is called after image (AFIM). In shadowing both BFIM and AFIM are maintained so the necessity of maintaining log in eliminated.

c) **Write-ahead Logging:** When in place updating is used, it is necessary to use a log for recovery. Here the recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on the disk. This process in known as write_ahead logging.

d) **Steal/No-Steal Approach:** In No-Steal approach a cache page updated by a transaction cannot be written to disk before the transaction commits. The pin-unpin bit is used to realize the approach. The bit indicates if a page cannot be written back to the disk. On the other hand, in Steal approach, a cache page updated by a transaction can be written to disk before the transaction commits.

e) **Force/No-Force Approach:** In Force approach all cache pages updated by a transaction are immediately written to disk when the transaction commits and otherwise it is called No-Force. A Typical Database systems use a Steal/No-Force strategy because of the following advantages:
   – Steal approach avoids the need for a very large buffer space to store all updated pages in memory.
   – No-Force approach provides the advantage of keeping an updated page of a committed transaction in the memory when another transaction needs to update it, thus it eliminates the I/O cost of reading that page again from disk.

f) **Check Point:** Check point is another type of entry that is maintained in the log. The recovery manager of a DBMS must decide at what intervals to put a check point. The interval may be measured in terms of time say every after ‘m’ second or minute or in the number ‘t’ of committed transactions since the last checkpoint. Here both ‘t’ or ‘m’ are system parameters. All transactions committed before a checkpoint entry do not need to redo their write operations.

Checkpoint creating process consists of the following actions:

a) Suspend execution of transactions temporarily.

b) Force-write all main memory buffers that have been modified to disk.

c) Write a checkpoint record to the log, and force-write the log to disk.

d) Resume executing transactions.

The time needed to force-write buffers to disk may delay transactions execution and can be a big performance issue.

g) Transaction Rollback: There may be the necessity of roll back the transaction in some situation when a transaction fails after updating the database. In case of transaction rollback the write operations are undone, and it must be restored to their previous values (BFIS).

Another variation of rollback known as cascading rollback is also there where if transaction T1 is rolled back, any Transaction T2 that has read the value of some data item X written by T1 must also be rolled back and so on.

To improve the performance, Fuzzy checkpoint is used to reduce the delay by resuming the execution of the transactions after the checkpoint record is written to the log without having to wait for force-write step to be completed. The previous checkpoint is valid until the force-write step is completed. This in done by the system by maintaining a pointer to the valid checkpoint and changes it to the new checkpoint after the success of the force-write step.



CHECK YOUR PROGRESS

Q.6: Fill up the gaps:

a) The information about the changes applied to the database item by various transactions are kept by the .....

b) Deferred update, updates the database only after the transaction is .....

c) In case of immediate update, the database is updated ..... a transaction is committed.

d) AFIM stands for .....

e) BFIM stands for .....

Q.7: State True or False:

a) Dirty bit indicates whether the buffer is modified or not.

b) In shadowing, the updated buffer is written at one location.

c) In case of Force Approach, database updates are written to the disk immediately.

d) Creation of checkpoint is the responsibility of recovery manager.

e) Rollback refers to restoring of the database to their previous value.



13.10 LET US SUM UP

- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved. There should be some mechanism in the system to control the interaction among the concurrent transactions.
- Concurrency control is the process of managing simultaneous execution of transactions in a multiprocessing database system without having them interfere with one another.
- This control mechanism enables many transactions to access the same database at the same time without interfering with each other.
- Concurrency control is important because the simultaneous execution of transactions over shared database can create several data integrity and consistency problem.
- In case of shared data, locking is a mechanism commonly used to solve the problem of synchronized access.

- Locks are used as means of synchronizing the access by concurrent transactions to the database items.
- Two-phase locking is a method or a protocol of controlling concurrent processing in which all locking operations precedes the first unlocking operations.
- Timestamp is a unique identifier created by the database management system to identify the relative starting time of a transaction.
- The main benefit of multiple version concurrency control is to help the scheduler avoid rejecting operations that arrive too late.



13.11 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.
2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



13.12 ANSWER TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) variable, b) lock conversion, c) Growing Phase, d) shrinking phase, e) aborts

Ans. to Q. No. 2: a) False, b) True

Ans. to Q. No. 3: a) updation, b) three, c) new

Ans. to Q. No. 4: a) True, b) False, c) True, d) False, e) True

Ans. to Q. No. 5: a) smallest, b) small, c) Coarse, d) transaction, e) multilevel

Ans. to Q. No. 6: a) system log, b) committed, c) before, d) After Image, e) Before Image

Ans. to Q. No. 7: a) True, b) False, c) True, d) True, e) True



13.13 MODEL QUESTION

Q.1: What is a lock? Explain the function of lock in concurrency.

Q.2: What is two phase locking protocol?

Q.3: Explain how two phase locking protocol guarantee serializability.

Q.4: What is a timestamp? Explain.

Q.5: What is certify lock? Explain the advantages and disadvantages of it.

Q.6: What is intention lock? Describe its different types.

Q.7: What do you mean by catastrophic failure?

Q.8: Describe the function of a Dirty bit.

*** ***** ***

UNIT 14: DATABASE RECOVERY

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 Database Recovery

14.4 Database Failure

14.5 Backup of Database for Catastrophic Failure

14.6 Recovery for Non-Catastrophic Failures

14.6.1 Deferred Update

14.6.2 Immediate Update

14.7 Let Us Sum Up

14.8 Further Reading

14.9 Answers to Check Your Progress

14.10 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- identify the types of database failure
- explain the types of database failure
- learn about database recovery
- learn to back up as database.

14.2 INTRODUCTION

We have already been acquainted with various important concepts related with database and DBMS. We have also learnt to insert, delete, update and other associated operations in a database through SQL commands. In this unit we will learn how a database can be recovered.

A database may become inconsistent because of a transaction failure, database system failure or media crash. Now in this case bringing the database into the last consistent state, which existed prior to the failure and to preserve transaction properties; recovery mechanism is a must.

In this unit we will learn about the types of failures, concept of database recovery and its need. At the end of this unit, we will discuss some of the techniques that can be used for database recovery from these failures.

14.3 DATABASE RECOVERY

Database recovery is the process of restoring the database to a consistent state that existed just before the time of failure. In this process, the information about the changes which are applied to data items carried out by various transactions are usually kept by the system. Typically this information is kept by the *system log*. System log files may contain information about device changes, device drivers, system changes, operations and more.

14.4 DATABASE FAILURES

A computer system, like any other device, is subject to failure from variety of causes. There are various types of failure that may occur in a system. Different failures are handled in different manner. In some failure loss of information does not occur. The failures that are more difficult to deal with are those that result in loss of information. Some failures affect the main memory only, while others involve secondary storage.

Failures are generally classified as catastrophic and non-catastrophic failure. System crashes, media failures etc. fall under catastrophic failure and transaction failure fall under non-catastrophic failure.

CATASTROPHIC FAILURE: In case of catastrophic failure like disk crash, there is an extensive damage to a wide portion of the database. In this condition, the recovery method restores a past copy of database that was backed up earlier to the archival storage and also reconstructs a more current state by reapplying and redoing the operations of committed transactions from the backed up log, to the time of failure. Some examples under this category: *disk head crash*, *power disruption*, *media failure* etc.

NON-CATASTROPHIC FAILURE: In case of non-catastrophic failure, the database on the disk is not physically damaged. Therefore, the main recovery strategy is to identify the changes that may cause any

inconsistency in the database. For example, a transaction that has updated some database items on disk but has not committed needs to have its changes reversed by undoing its write operation. It may also be necessary to redo some operations in order to restore a consistent state of the database. Here, a complete archival copy of database is not necessarily needed by the recovery protocol. Rather, to determine the appropriate actions for recovery, the entries kept in online system log on disk are analyzed. Some examples: Transaction error due to incorrect input, deadlock, incorrect synchronization etc.

Let us now discuss briefly discuss different types of failures:

1) **System Crashes:** System crashes are due to hardware malfunction or software errors or the operating system failure, resulting in the loss of the content of main memory. The following failures can be included in system crash.
   - **Hardware failures:** Hardware failures may include disk full errors, memory errors, disk crashes, bad disk sectors and so on.
   - **Software failures:** Software failure generally includes operating system failure, DBMS failure, application program failure and so on i.e., failures related to software.
   - **Network failures:** Network failure usually occurs while using a client-s erver system or a distributed system where multiple servers are connected by common networks.
2) **Transaction Failure:** A transaction may fail due to *logical error* and s*ystem error.*
   - **Local errors:** Sometime while executing a transaction, certain conditions may occur that necessitate cancellation of the transaction. Some examples of logical errors are: wrong input, overflow, indufficient data for the transaction etc. For instance, some exceptional situation may arise such as insuffiecient account balance in a banking database that may cause a transaction like fund withdrawal to be cancelled.
   - **System errors:** Normal execution of a transaction may be inturrupted as the system enters an undesirable state such as

deadlock. The transaction, however, can be reexecuted. Such type of errors fall under system error.

3) **Media Failures:** Media failures are most dangerous. It occurs due to head crashes or unreadable media which results in loss of parts of secondary storage. They are the most dangerous failure.
4) **Natural Disaster Failures:** We may sometime lose our databases due to natural disaster such as fires, power failures, floods or earthquake.
5) **Carelessness:** Sometime we may lose our important databases due to our carelessness or carelessness of users or operators.
6) **Sabotage:** These failures occur due to intentional corruption or destruction of data, hardware, or software facilities.

14.5 BACKUP OF DATABASE FOR CATASTROPHIC FAILURE

In catastrophic failure, the main technique used is a database back up. In this process, the whole database and the log is periodically copied to a storage medium like magnetic tapes or other large capacity offline storage devices. In case of catastrophic system failure, the latest back up copy can be reloaded from the tape to the disk, and the system can be restarted. Data from critical applications such as banking, insurance, stock market, and other databases are periodically backed up and moved to a physically safe separate locations.

To avoid loss of effects of transactions that have been executed since the last back up, it is mandatory to back up the system log at more frequent intervals than the full database back up by periodically copying it to magnetic tape. The system log can be backed up more frequently as it is usually smaller than the database. Therefore, the users do not have to lose all the transactions being performed.



CHECK YOUR PROGRESS

Q.1: Discuss the different types of transaction failures. What is meant by catastrophic failure?

Q.2: How are the catastrophic failure and non-catastrophic failure recovery techniques different?

Q.3: Discuss how disaster recovery from catastrophic failures is handled.

14.6 RECOVERY FOR NON-CATASTROPHIC FAILURES

Recovery mechanism from Non-catastrophic transaction failures are of two types: *deferred update* and *immediate update*.

14.6.1 Deferred Update

In deferred update strategy, database is modified only after reaching the commit point of the transactions. The idea behind deferred update is to postpone any actual updates to the database on disk until the transaction reach its commit point by executing successfully. During the execution of the transaction, the updates are recorded only in the log and in the cache buffers. When the transaction reaches its commit point and the log is forced written to the disk, the updates are recorded in the database. If before reaching its commit point the transaction fails, there is no need to undo any operations as the transaction has not affected the database on disk in any way. We can summerized the deferred update recovery in the following steps:

- A transaction cannot change the database on disk until the commit point is achieved.
- A transaction cannot reach the commit point until all its update operations are written onto log and the log is force written to disk.

14.6.2 Immediate Update

In this technique, when a transaction issues an update command, immediately the database on disk can be updated. Here, it is not required to wait for the transaction to reach the commit point. Provisions must be made for undoing the effect of updated operations that have been applied by the failed transactions to the database. This is performed by rolling back the transaction and undoing the effect of the transaction's write item operations. Therefore, the UNDO-type log entries that include the old value of the items must be stored in the log. Two main types of immediate update algorithms are:

- UNDO/NO-REDO recovery algorithm
- UNDO/REDO recovery algorithm

UNDO/NO-REDO Recovery Algorithm: The recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is no need to REDO any operations of committed transactions ever. In this method, all the updates by the transaction must be recorded on disk before reaching the commit point, so that REDO is never needed. It must utilize the force strategy for deciding when update main memory buffers are written back to disk.

UNDO/REDO Recovery Algorithm: In this technique the transaction is allowed to commit before all its changes are written to the database.



CHECK YOUR PROGRESS

Q.4: What is the difference between the UNDO/ REDO and the UNDO/NO-REDO algorithms for recovery with immediate update?



14.7 LET US SUM UP

In this chapter we have discussed about various types of database failures: catastrophic failures and non-catastrophic failure. The database

may lose transaction consistency because of the occurrence of these failures. Because of this various recovery techniques need to be implemented. Here we have discussed various database recovery techniques for this very purpose.



14.8 FURTHER READING

1) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
2) Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems*. McGraw Hill.
3) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.



14.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: Catastrophic and Non-Catastrophic Failure. If there is extensive damage to a wide portion of the database then it is called Catastrophic Failure.

Ans. to Q. No. 2: In catastrophic failure, there is extensive damage to a wide portion of the database; while if a non-catastrophic failure has occurred then the database on disk is not physically damaged.

Ans. to Q. No. 3: In case of a catastrophic failure like such as disk crashes, the main technique used is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices.

Ans. to Q. No. 4: In Undo/Redo technique the transaction is allowed to commit before all its changes are written to the database; while in Undo/No-Redo method, all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed.



14.10 MODEL QUESTIONS

Q.1: Discuss the actions taken by the read_item and write_item operations on a database.

Q.2: What is the system log used for? What are the typical kinds of entries in a system log? What are checkpoints, and why are they important? What are transaction commit points, and why are they important?

Q.3: How are buffering and caching techniques used by the recovery subsystem?

Q.4: What are the Before image (BFIM) and After image (AFIM) of a data item? What is the difference between in-place updating and shadowing, with respect to their handling of BFIM and AFIM?

Q.5: What are UNDO-type and REDO-type log entries?

Q.6: Describe the write-ahead logging protocol.

Q.7: Identify three typical lists of transactions that are maintained by the recovery subsystem.

Q.8: What is meant by transaction rollback? What is meant by cascading rollback? Why do practical recovery methods use protocols that do not permit cascading rollback? Which recovery techniques do not require any rollback?

Q.9: Discuss the UNDO and REDO operations and the recovery techniques that use each.

Q.10: How can recovery handle transaction operations that do not affect the database, such as the printing of reports by a transaction?

Q.11: Discuss the immediate update recovery technique in both single-user and multiuser environments. What are the advantages and disadvantages of immediate update?

Q.12: Describe the shadow paging recovery technique. Under what circumstances does it not require a log?

*** ***** ***

UNIT 15: SECURITY AND AUTHENTICATION

UNIT STRUCTURE

15.1 Learning Objectives

15.2 Introduction

15.3 Database Security Issue

15.4 Discretionary Access Control Based On Granting and Revoking Privileges

15.5 Mandatory Access Control

15.6 Role Based Access Control

15.7 Encryption and Public Key Infrastructures

15.8 Let Us Sum Up

15.9 Further Reading

15.10 Answers to Check Your Progress

15.11 Model Questions

15.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe database security
- define discretionary access control
- describe mandatory access control
- describe role based access control
- describe encryption and public key infrastructures.

15.2 INTRODUCTION

We have already learned various concepts and techniques of database recovery in the previous unit. This unit discusses the techniques used for protecting the database against the person who are not authorized to access either certain part of database or the whole database.

15.3 DATABASE SECURITY ISSUE

Database security is a very broad area that addresses many issues, including the following:

- Legal and ethical issue regarding the right to access certain information. Sometime information may be deemed to be private and cannot be accessed legally by unauthorized persons.
- Policy issue at the governmental, institutional, or corporate level as to what kind of information should not be made publicly available- for example credit rating and personal medical record.
- System related issues such as the system levels at which various security functions should be enforced. For example, where a security functions should be handled at the physical hardware level, the operating system level or the DBMS level.
- The need in some organizations to identify multiple security levels and to categorize the data users based on these classifications; for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications must be enforced.

Threats to Databases: Threat to database results in the loss or degradation of some or all of the following security goals: integrity, availability, and confidentiality.

- **Loss of integrity:** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to data by either intentional or accidental acts. If the loss of the system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decision.
- **Loss of availability:** Database availability refers to making objects available to human user or a program to which they have a legitimate right.
- **Loss of confidentiality:** Database confidentiality refers to the protection of data from unauthorized disclosure.

Database Security and the DBA: The database administrator (DBA) is the central authority for managing a database system. The DBA's

responsibilities include granting privileges to the users who need to use the system and classifying users and data in accordance with the policy of the organization. The DBA has a DBA account in the DBMS, sometimes called a system or super user account, which provides powerful capabilities. The DBA is responsible for the overall security of the database system.

Access Protection, User Accounts, and Database Audits: Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database. The user must **log in** to the DBMS by entering account number and password whenever database access is needed.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each **login session**.

To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify ***system log***, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.

If any tampering with the database is suspected, a **database audit** is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. A database log that is used mainly for security purposes is sometimes called an **audit trail**.



CHECK YOUR PROGRESS

Q.1: Choose the correct answer:

i) Modification of data includes–

a) Creation b) Insertion

c) Modification d) All of the above

ii) DBA stands for–

a) Database Access b) Database Administrator

c) Data Bound Administrator d) None of the above

iii) A database log that is used mainly for security purposes is sometimes called–

a) System log b) Database Audit

c) Database trial d) Audit trial

15.4 DISCRETIONARY ACCESS CONTROL BASED ON GRANTING AND REVOKING PRIVILEGES

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.

There are two levels for assigning privileges:

- **The account level:** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- **The relation level:** At this level, the DBA can control the privilege to access each individual relation or view in the database.

The privileges at the account level apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege, to apply schema changes such as adding or removing attributes from the relation; the DROP privilege, to delete relations or view; the MODIFY privilege, to insert, delete, or update tuples, and the SELECT privilege, to retrieve information from the database using a SELECT query. Notice that these account privileges apply to the account in general. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account.

The second level of privileges applies to the relational level, whether they are base relations or virtual relations.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an owner account, which is typically the account that was used when the relation was created in the first place. The owner relation is given all the privileges on that relation.

In SQL the following types of privileges can be granted to each individual relation R:

- SELECT privilege on R: Gives the account retrieval privilege.
- MODIFY privileges on R: This gives the account the capability to modify the tuples of R.
- REFERENCE privilege on R: This gives the account the capability to reference relation R when specifying integrity constraints. This privilege can also restrict specific attributes of R.

The mechanism of views is an important discretionary authorization mechanism in its own right. For example if the owner A of relation R wants another account B to be able to retrieve only some field of R, then A can create a view V of R that includes only those attribute and then grant SELECT on V to B.

Revoking Privilege: In some case it is desirable to grant privilege to a user temporarily. For example, the owner of relation may want to grant the SELECT privilege to a user for a specific task and revoke the privilege once the task is completed. Hence, a mechanism for revoking privileges is needed. In SQL REVOKE command is included for the purpose of cancelling privileges.

Propagation of Privileges using the GRANT OPTION: Whenever the owner A of relation R grants privilege on R to another account B, the privilege can be given to B with or without the GRANT OPTION. If the GRANT OPTION is given, this means that B can also grant privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B grants the privilege on R to a third account C, also with GRANT OPTION. In this way, privilege on R can propagate to other account without the knowledge of the owner of R. If the owner account A now revoke the privilege granted to B, all the privilege that B propagate based on that privilege should automatically be revoked by the system.

Let us consider an example. Suppose that the DBA creates four accounts A1, A2, A3, and A4 and wants only A1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB TO A1;

In SQL2 the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command as follows:

CREATE SCHAMA EXAMPLE AUTHORIZATION A1;

Now user account A1 can create tables under the schema called EXAMPLE.

Suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT; A1 is then owner of these two relations and hence all the relation privileges on each of them.

Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

DEPARTMENT

DNUMBER	DNAME	MGRSSN

Suppose that A1 wants to allow A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts.

A1 can issue the command:

GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION;

A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing:

GRANT SELECT ON EMPLOYEE TO A4;

(Notice that A4 cannot propagate the SELECT privilege because GRANT OPTION was not given to A4.)

Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 can issue:

REVOKE SELECT ON EMPLOYEE FROM A3;

(The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.)

Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.

A1 then create the view:

```
CREATE VIEW A3EMPLOYEE AS
SELECT NAME, BDATE, ADDRESS
FROM EMPLOYEE
WHERE DNO = 5;
```

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;
```

Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE;

A1 can issue:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO A4;
```

(The UPDATE or INSERT privilege can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute specific.)



CHECK YOUR PROGRESS

Q.2: Choose the correct option:

i) To create a schema or base relation ..... privilege used.

a) Create table b) Drop

c) Select d) Delete

ii) To apply schema changes such as adding or removing attributes from the relation ..... privilege used.

a) Alter b) Modify

c) Delete d) Non of the above

iii) To insert, delete, or update tuples ..... privilege used.

a) Alter b) Modify

c) Delete d) Insert

iv) ..... command is used for the purpose of cancelling privileges

a) Grant b) Revoke

c) Delete d) Non of the above

15.5 MANDATORY ACCESS CONTROL

It is a security policy that classifies data and the users based on security class. Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is higher level and U is lower level. Other security classifications exist, in which the security classes are organized in a lattice. The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each subject (user, account, programs) and object (relation, tuple, column, view, operation) in to one of the security classifications TS, S, C, or U. We will refer to the clearance (classification) of a subject S as **class(S)** and to the classification of an object O as **class (O)**. Two restrictions are enforced on data access based on the subject/ object classifications:

1) A subject S is not allowed read access to an object O unless class(S) e" class(O). This is known as the simple security property.
2) A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as star property (or *-property).

Mandatory policy ensures a high degree of protection- in a way, it prevents any illegal flow of information. They are therefore suitable for military types of applications, which require a high degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict classification of subjects and objects into security levels and, therefore, they are applicable to very few environments.

To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects.

Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. Hence, a **multilevel relation** schema R with n attributes would be represented as:

$$R(A_1,C_1,A_2,C_2, \ldots, A_n,C_n,TC)$$

Where, each C_i represents the classification attribute associated with attribute A_i.

The value of the TC attribute in each tuple t –which is the *highest* of all attribute classification values within t –provides a general classification for the tuple itself, whereas each C_i provides a finer security classification for each attribute value within the tuple.

The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.

A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**.

In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the *apparent key*. This leads to the concept of **polyinstantiation** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple.

In addition, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key. This constraint ensures that a user can see the key if the user is permitted to see any part of the tuple at all.

Other integrity rules, called **null integrity** and **interinstance integrity**, informally ensure that if a tuple value at some security level can be filtered

(derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.

15.6 ROLE BASED ACCESS CONTROL

Role-based access control (RBAC) emerged rapidly in 1990s as a proven technology for managing and enforcing security in large scale enterprisewide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands can be used to assigned and revoke privileges from role.

RBAC ensures that only authorize user can access to the certain data or resources. Users create session during which they may active a subset of roles to which they belong. Each session can be assigned to many roles, but it maps to only one user or a single subject.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.

Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as time and duration of role activations, and timed triggering of a role by an activation of another role.

Using an RBAC model is highly desirable goal for addressing the key security requirements of Web-based applications.

15.7 ENCRYPTION AND PUBLIC KEY INFRASTRUCTURES

Encryption is a means of maintaining secure data in an insecure environment. Encryption consists of applying an **encryption algorithm** to data using some prespecified **encryption key**. The resulting data has to be **decrypted** using a **decryption key** to recover the original data.

In 1976 Diffie and Hellman proposed a new kind of cryptosystem, which they called **public key encryption**. Public key algorithms are based on mathematical functions rather than operations on bit patterns. They also

involve the use of two separate keys, in contrast to conventional encryption, which uses only one key. The use of two keys can have profound consequences in the areas of confidentiality, key distribution, and authentication. The two keys used for public key encryption are referred to as the **public key** and the **private key**. The private key is kept secret, but it is referred to as *private key* rather than a *secret key* (the key used in conventional encryption) to avoid confusion with conventional encryption.

A public key encryption scheme, or infrastructure, has six ingredients:

1) *Plaintext:* This is the data or readable message that is fed into the algorithm as input.
2) *Encryption algorithm:* The encryption algorithm performs various transformations on the plaintext.
3) and 4) *Public and private keys:* These are pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exect transformations performed by the encryption algorithm depend on the public or private key that is provided as input.
5) *Ciphertext:* This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
6) *Decryption algorithm:* This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public key is made for public and private key is known only by owner.

A general-purpose public key cryptographic algorithm relies on one key for encryption and a different but related one for decryption. The essential steps are as follows:

1) Each user generates a pair of keys to be used for the encryption and decryption of messages.
2) Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.
3) If a sender wishes to send a private message to a receiver, the sender encrypts the message using the receiver's public key.

4) When the receiver receives the message, he or she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his or her private key.

The RSA Public Key Encryption Algorithm, one of the first public key schemes was introduced in 1978 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and is named after them as the RSA scheme. The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target. The RSA algorithm also operates with modular arithmetic– mod n.

Two keys, *d* and *e*, are used for decryption and encryption. An important property is that d and e can be interchanged. *n* is chosen as a large integer that is a product of two large distinct prime numbers, *a* and *b*. The encryption key *e* is a randomly chosen number between *1* and *n* that is relatively prime to *(a–1) x (b–1).* The plaintext block P is encrypted as P^e *mod n.* Because the exponentiation is performed *mod n,* factoring P^e to uncover the encrypted plaintext is difficult. However, the decryption key *d* is carefully chosen so that $(P^e)^d$ *mod n* = *P*. The decryption key *d* can be computed from the condition that *d x e= 1 mod ((a–1)x(b–1)).*Thus, the legitimate receiver who knows *d* simply computes $(P^e)^d$ *mod n* = *P* and recovers *P* without having to factor P^e.



CHECK YOUR PROGRESS

Q.3: Choose the correct option:

i) Typical security classes are top secret (TS), secret (S), confidential (C), and .....

a) Simple b) Unclassified

c) Classified d) None of the above

ii) A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as:

a) Star property b) Close property

c) Simple Access d) None of the above

iii) RBAC stand for–

a) Role-based access control

b) Role-based action control

c) Role-based access concurrency

d) None of the above

iv) ..... is a means of maintaining secure data in an insecure environment.

a) Encryption b) Decryption

c) Encoding d) Decoding



15.8 LET US SUM UP

- Database security issues are legal and ethical issue, policy issue, system related issues.
- Threat to database result in the loss or degradation of integrity, availability, and confidentiality.
- The database administrator (DBA) is the central authority for managing a database system.
- If any tampering with the database is suspected, a **database audit** is performed.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.
- The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.
- In SQL REVOKE command is included for the purpose of cancelling privileges.
- Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U).
- A subject S is not allowed read access to an object O unless class(S) e" class(O). This is known as the simple security property.
- A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as star property (or *-property).

- The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.
- Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.
- Encryption is a means of maintaining secure data in an insecure environment.
- *Public and private keys*: These are pair of keys that have been selected so that if one is used for encryption, the other is used for decryption.



15.9 FURTHER READING

1) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
2) Leon, A. & Leon, M. *Fundamentals of Database Management System*. Tata McGraw Hill.



15.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (d), ii) (b), iii) (d)

Ans. to Q. No. 2: i) (a), ii) (a), iii) (b), iv) (b)

Ans. to Q. No. 3: i) (b), ii) (a), iii) (a), iv) (a)



15.11 MODEL QUESTIONS

Q.1: What is database security?

Q.2: What are the different database security issues?

Q.3: What do you mean by Loss of Integrity?

Q.4: What is meant by granting a privilege?

Q.5: What is meant by revoking a privilege?

Q.6: List the type of privilege available in SQL.

Q.7: What is the difference between discretionary and mandatory access control?

Q.8: Define the following terms: apparent key, polyinstantiation, filtering.

Q.9: What are the relative merits of using DAC and MAC?

Q.10: What is role-based access control? In what ways is it superior to DAC and MAC?

Q.11: What is the goal of encryption? What process is involved in encrypting data and then recovering it at the other end?

Q.12: What is public key infrastructure? How does it provide security?

Q.13: Give an example of an encryption algorithm and explain how it works.

*** ***** ***



REFERENCES

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
3) Leon, A. & Leon, M. *Fundamentals of Database Management System*. Tata McGraw Hill.
4) Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems*. McGraw Hill.
5) Silberschatz, A., Korth, H. F. & Sudarshan, S. (1997). *Database System Concepts* (Vol. 4). New York: McGraw-Hill.
6) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.
7) Ullman, J. D. (1984). *Principles of Database Systems*. Galgotia Publications.

Name of the Paper: .....
Course Code: .....

Centre for Internal Quality Assurance (CIQA)
Krishna Kanta Handiqui State Open University
City Office: Housefed Complex, Guwahati-781006

Learner's Feedback on Course

Dear Learner,

Regarding the course as mentioned above, we would like to know your opinions and comments so as to improve the quality of self learning materials in future. Please respond to the following statements by ticking the number you feel most reflect your opinion. After completion of the additional comments, please detach the page and send/mail the same to us at the address given below.

The Director, Centre for Internal Quality Assurance, KKHandiqui State Open University
Housefed Complex, Dispur, Guwahati-781006
(E-mail id: ciqa@kkhsou.in)

1) Approximately how many hours did you spend for studying the units in the course? ☐☐☐

2) Please give your opinions (by ✓ mark) to the following items based on your reading of the block:

Sl. No.	Statements	Strongly Agree	Agree	Neither Agree nor Disagree	Disagree	Strongly Disagree
I)	The SLMs of the course fulfil the learning objectives set out in the beginning of each unit					
II)	The units of the SLM could satisfy my academic needs and expectations					
III)	The Unit writers have excellent knowledge about the course contents					
IV)	Language and contents of the units were presented at a level which I could readily understand					
V)	Ample opportunity for participation in the activities provided in the units					
VI)	Used enough Illustrations (Diagrams, tables etc.) for conceptual clarity					
VII)	Quality of content is engaging, relevant, and up-to-date					
VIII)	The self check questions are very helpful					
IX)	The Possible/model questions and the answers to check my progress have benefited me a lot					

Additional Comments: *(Please feel free to provide your open comments)*

1) Which aspects of the SLM, according to you, worked well?

   .....

   .....

2) What sort of changes/improvements do you feel KKHSOU could implement to improve the overall quality of the SLM?

   .....

   .....

Thank you for taking the time to complete this form.