CHAPTER 3

RELATIONAL DATA MODEL AND RELATIONAL DATA BASE CONSTRAINTS

OBJECTIVES

3.0. Introduction - Relational Model
3.1. Use Relational Model Concepts
3.2. Describe Relational Model Constraints
3.3. Illustrate Relational Database schema
3.4. Describe Update operation and dealing with constraints violations
3.5. Define Transaction

3.0. Introduction - Relational Model

3.0.1. What is Relational Model?

- **Relational Model (Rm)** represents the database as a collection of relations.
- A relation is nothing but a table of values. Every row in the table represents a collection of related data values.
- These rows in the table denote a real-world entity or relationship.
- The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations.
- In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.
- Some popular Relational Database management systems are:
  - ❖ DB2 and Informix Dynamic Server - IBM
  - ❖ Oracle and RDB – Oracle
  - ❖ SQL Server and Access - Microsoft

3.0.2. Advantages of using Relational model

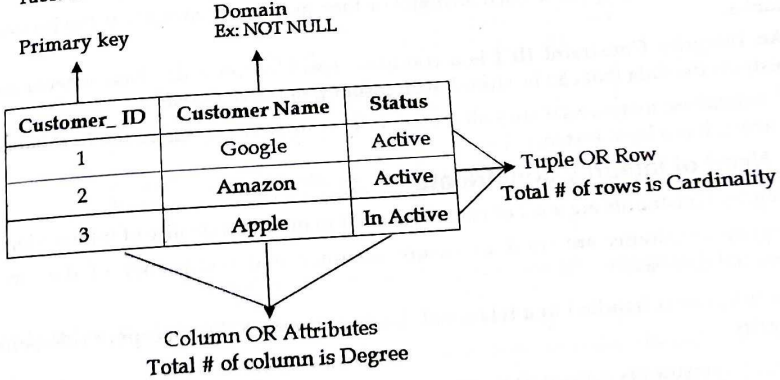
1. **Simplicity:** A relational data model is simple than the hierarchical and network model.
2. **Structural independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
3. **Easy to use:** The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
4. **Query capability:** It makes possible for a high-level query language like SQL to avoid complex database navigation.
5. **Data independence:** The structure of a database can be changed without having to change any application.
6. **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

3.0.3. Disadvantages of using Relational model

1. Few relational databases have limits on field lengths which can't be exceeded.
2. Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
3. Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

3.1 Use Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. **Tables:** In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple:** It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance:** Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key:** Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain:** Every attribute has some pre-defined value and scope which is known as attribute domain



3.1.1. Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name

- Attribute domain has no significance
- Tuple has no duplicate value
- Order of tuple can have a different sequence

Note:

- Data need to be represented as a collection of relations.
- Each relation should be depicted clearly in the table.
- Rows should contain data about instances of an entity.
- Columns must contain data about attributes of the entity.
- Cells of the table should hold a single value.
- Each column should be given a unique name.
- No two rows can be identical.
- The values of an attributes should be from the same domain.

3.2 Describe Relational Model Constraints

- While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints.
- These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.

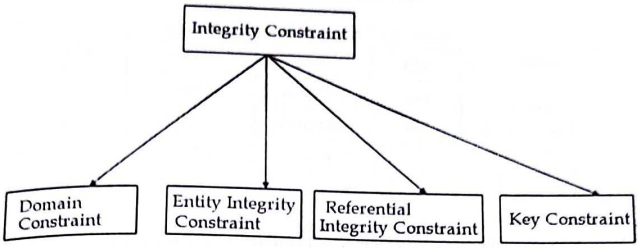
3.2.1. Integrity Constraints over Relations

- Constraints may apply to each attribute or they may apply to relationship between tables.
- **An Integrity Constraint (IC)** is a condition specified on a database schema and restricts the data that can be stored in an instance of the database.
- If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance.

3.2.2. Need of integrity constraints

- Integrity constraints are a set of rules. It is used to maintain quality of information.
- Integrity constrains are used to ensure accuracy and consistency of data in a relational database.
- Data integrity is handled in a relational database through the concept of **referential integrity.**
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Integrity constraints is used to guard against accidental damage to the database.

3.2.3. Types of Integrity Constraints



a) Domain Constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc.
- The value of the attribute must be available in the corresponding domain.
- **Example:**

ID	Name	Semester	Age
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

b) Entity Integrity Constraints

- The entity integrity constraints states that primary key value can't be null.
- Because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

- **Example 1:**

EMPLOYEE

EMP_ID	EMP_Name	Salary
123	Tom	30000
142	Johnson	40000
164	Leonardo	20000
	Kate	20035

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

- **Example 2:** Consider a relation "STUDENT" where "Stu_id" is a primary key and it must not contain any null value whereas other attributes may contain null value. Here, "Branch" in the following relation contains one null value.

Stu_Id	Name	Branch
11255234	Aman	EEE
11255369	Kapil	ECE
11255324	Ajay	
11255367	Raman	CSE

c) Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary key of Table 2, then every value of the Foreign key in Table 1 must be null or be available in Table 2.
- **Rules**
  - ❖ You can't delete a record from a primary table if matching records exist in a related table.
  - ❖ You can't change a primary key value in the primary table if that record has related records.
  - ❖ You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
  - ❖ However, we can enter a Null value in the foreign key, specifying that the records are unrelated.

- **Example 1:**

Emp_ Name
1
2
3
4

Pr

d) Key Constrai

- Keys are the uniquely.
- An entity set key.
- A primary ke
- **Example:**

- **Example 1:**

(Table 1)

Emp_ Name	Name	Age	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	James	38	13

Foreign Key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

Primary Key

D_No	D_Loc
11	Mumbai
24	Delhi
13	Noida

d) Key Constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key.
- A primary key can contain a unique and null value in the relational table.
- **Example:**

ID	Name	Semester	Age
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because all rows must be Unique

3.3 Illustrate Relational Database schema

- A relational database schema is an arrangement of integrity constraints.
- The integrity constraints that are specified on database schema shall apply to every database state of that schema.
- Let us consider a relational database schema COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.
- The underlined attributes represent primary keys as shown below.

EMPLOYEE

Fname	Minit	Lname	SSN	Bdate	Address	Salary	Super_SSN	Dno

DEPARTMENT

Dname	Dnumber	Mge_SSN	Mgr_start_date

DEPARTMENT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

ESSN	Pno	Hours

DEPENDANT

ESSN	Dependent_name	Gender	Bdate	Relationship

Schema Diagram for the COMPANY Relational Database Schema

- Points to be considered in designing the Schema are
  - Attributes that represent the same real-world concept may or may not have identical names in different relations.
  - For example in the above figure the Dnumber attribute in both DEPARTMENT and DEPT_LOCATIONS as well as Dno in EMPLOYEE and Dnum in PROJECT represents same real world.
  - Attributes that represent different real-world concepts may have the same name in different relations.
  - For example, we could have used the attribute-name Name for both Pname of PROJECT and Dname of DEPARTMENT; in this case, we would have two attributes that share the same name but represent different real-world concepts – project names and department names.

- Use Different attribute names when a single attribute is used for different roles in same relation.
- For example the concept of Social Security number appears twice in the EMPLOYEE relation in the above figure , once in the role of the employee's SSN, and once in the role of the super-visor's SSN.
- Each relational DBMS must have a data definition language (DDL) for defining a relational database schema. Current relational DBMSs are mostly using SQL for this purpose.

3.4 Describe Update operation and dealing with constraints violations

- There are mainly Four basic update operations performed on relational database model are Insert, update, delete and select.
  - **Select:** It allows you to choose a specific range of data.
  - **Insert:** It is used to insert data into the relation
  - **Update:** It is used to delete tuples from the table.
  - **Delete:** It allows you to change the values of some attributes in existing tuples.
- Whenever one of these operations are applied, integrity constrains specified on the relational database schema must never be violated.

3.4.1. SELECT Operation

- SELECT statement retrieves data from the database.
- The data is returned in a table-like structure called a result-set.
- SELECT is the most frequently used action on a database.
- **Syntax:**

  SELECT (Column_name1, Column_name2, Column_name3, ...) From table_name;
- **Example:** Consider the following student table

s_id	Name	Age	Address
101	Adam	15	Chennai
102	Alex	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

The above query will fetch information of s_id, name and age columns of the student table and display them.

s_id	Name	Age
101	Adam	15
102	Alex	18
103	Abhi	17
104	Ankit	22

3.4.2. INSERT Operation

- The INSERT Operation is used to insert new records into the table.
- The INSERT operation gives values of the attributes(columns) for a new tuple(row) which should be inserted into a relation.
- **Syntax:**

```
INSERT INTO table_name VALUES (value1, value2,.....)
```

- **Example:** Consider a table **STUDENT** with the following attributes

s_id	name	Age

```
INSERT INTO STUDENT VALUES (101, 'Adam' , 15);
```

The above command will insert a new record into student table.

s_id	name	age
101	Adam	15

a) INSERT values into only specific columns

- INSERT command can also be used to insert values for only some specific columns of a row. We can specify the column names along with the values to be inserted.
- **Syntax:**

```
INSERT INTO table_name (column1, column2, column3,....) VALUES (value1, value2, value 3) ;
```

- **Example:**

```
INSERT INTO STUDENT (id, name) VALUES (102, 'Alex') ;
```

- The above SQL query will only insert id and name values in the newly inserted record.

b) INSERT NULL value to a column

- Both the statements below will insert NULL value into age column of the student table.

```
INSERT INTO STUDENT (id, name ) VALUES (102, 'Alex');
```

Or,

```
INSERT INTO STUDENT VALUES (102, 'Alex', null);
```

- The above command will insert only two column values and the other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

c) INSERT Default values to a column

```
INSERT INTO STUDENT VALUES (103, 'Chris', default)
```

s_id	S_name	age
101	Adam	15
102	Alex	
103	Chris	14

3.4.3. UPDATE Operation

- UPDATE statement is used to modify the existing records in a table.
- **Syntax:**

```
UPDATE table_name SET Column_name = New_vlaue WHERE some_conditon;
```

- The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated.
- **Example:** Let's take as sample table **STUDENT**,

Student_id	name	age
101	Adam	15
102	Alex	
103	chris	14

```
UPDATE student SET age=18 WHERE student_id=102;
```

Student_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

3.4.4. DELETE Operation

- The DELETE operation is used to delete existing records in a table.
- The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

- **Syntax:**

```
DELETE FROM table_name WHERE condition;
```

- **Example:** Let's take a sample table STUDENT

s_id	name	age
101	Adam	15
102	Alex	18
103	Abhi	17

```
DELETE FROM student WHERE name='Alex';
```

s_id	Name	Age
101	Adam	15
102	Alex	18

a) Delete all Records from a Table

- **Syntax:**

```
DELETE FROM table_name ;
```

- **Example:**

```
DELETE FROM STUDENT ;
```

- The above command will delete all the records from the table student

3.5 Define Transaction

3.5.1. Transaction

- A transaction can be defined as a group of tasks that form a single logical unit.
- For example, Suppose we want to withdraw Rs. 100 from an account then we will follow following operations :
  - ❖ Check account balance
  - ❖ If sufficient balance is present request for withdrawal.
  - ❖ Get the money
  - ❖ Calculate Balance = Balance -100
  - ❖ Update account with new balance.
- The above mentioned four steps denote one transaction
- In database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

3.5.2. ACID Properties

a) Atomicity

- This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.
- No transaction in the database is left half completed.
- Data should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.
- For example, In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e. customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

b) Consistency

- The database must remain in consistent state after performing any transaction.
- For example, In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

c) Isolation

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transaction will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- For example, If a bank manger is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

d) Durability

- The database should be strong enough to handle any system failure.
- If there is any set of inset/update, then it should be able to handle and commit to the database.
- If there is any failure, the database should be able to recover it to the consistent state.

- For example, In ATM withdrawal example, if the system failure happens after Customer getting the money then the system should be strong enough to update Database with his new balance, after system recovers. For that purpose the system has to keep the log of each transaction and its failure. So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to Database.

4.0. Introduction

4.0.1. What is Normalization ?

1. Normalization is the process for assigning attributes to entities. Properly executed, the normalization process eliminates uncontrolled data redundancies; thus eliminating the data anomalies and the data integrity problems that are produced by such redundancies.
2. Normalization does not eliminate data redundancy; instead, it produces the carefully controlled redundancy that lets us properly link database tables.
3. In other words we can say that **Database normalization** is a design technique by which relational database tables are structured in such a way as to make them invulnerable to certain types of logical inconsistencies and anomalies.
4. Tables can be normalized to varying degrees: relational database theory defines "normal forms" of successively higher degrees of stringency, so, for example, a table in third normal form is less open to logical inconsistencies and anomalies that a table that is only in second normal form.
5. Although the normal forms are often defined (informally) in terms of the characteristics of tables, rigorous definitions of the normal forms are concerned with the characteristics of mathematical constructs known as relations.
6. Whenever information is represented relationally-that is, roughly speaking, as values within rows beneath fixed column headings-it makes sense to ask to what extent the representation is normalized.

4.0.2. Need of Normalization

1. Normalization is the process of reorganizing data in a database so that it meets two basic requirements:
   - There is **no redundancy** of data (all data is stored in only one place), and
   - Data **dependencies** are logical (all related data items are stored together)
2. The normalization is important because it allows database to take up **less disk space.**
3. It also help in increasing the **performance.**

4.0.3. Data Redundancy

1. Data redundancy is a data organization issue that allows the unnecessary duplication of data within your database.
2. A change or modification, to redundant data, requires that you make changes to multiple fields of a database. While this is the expected behaviour for flat file database designs and spreadsheets, it defeats the purpose of relational database designs.

3. The data relationships, inherent in a relational database, should allow you to maintain a single data field, at one location, and make the database's relational model responsible to port any changes, to that data field, across the database.
4. Redundant data wastes valuable space and creates troubling database maintenance problems.
5. Data should not be redundant, which means that the duplication of data should be kept to a minimum for several reasons. For example, it is unnecessary to store an employee's home address in more than one table.
6. To eliminate redundant data from your database, you must take special care to organize the data in your tables.
7. Normalization is a method of organizing your data to prevent redundancy. Normalization involves establishing and maintaining the integrity of your data tables as well as eliminating inconsistent data dependencies.
8. Establishing and maintaining integrity requires that you follow the database prescribed rules to maintain parent-child, table relationships.
9. Eliminating inconsistent, data dependencies involve ensuring that data is housed in the appropriate database table. An appropriate table is a table in which the data has some relation to or dependence on the table.
10. Normalization requires that you adhere to rules, established by the database community, to ensure that data is organized efficiently. These rules are called normal form rules.
11. Normalization may require that you include additional data tables in your Access database. Normal form rules number from one to three, for most applications.

4.1 Explain Informal Design guidelines for relation schemas

- The four informal Guidelines to measures the quality of relational schema design are:
  - **Guideline 1:** Semantic of Relation Attributes
  - **Guideline 2:** Reducing the Redundant values in Tuples
  - **Guideline 3:** Reducing the Null Values in Tuples
  - **Guideline 4:** Disallowing Generating Spurious Tuples

4.1.1. Guideline 1: Semantic of Relation Attributes

- Design a relational schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- If a relation schema corresponds to one entity type, or one relationship type, the meaning tends to be clear

a) Example of Good Design

EMPLOYEE

Ename	SSN	Ddate	Address	Dnumber
Salah	111	1965-01-09	Khobar	5
Wael	222	1955-12-08	khobar	5
Zaher	333	1968-07-19	Dammam	4
Walid	444	1941-06-20	Dhahran	4
Nasser	555	1962-09-15	Jubail	5
Essam	666	1972-07-31	Khobar	5
Jabber	777	1969-03-29	Khobar	4
Bader	888	1937-11-10	Khobar	1

DEPARTMENT

Dname	Dnumber	DMGRManager
Research	5	222
Administration	4	444
Headquarters	1	888

PROJECT

Pname	Pnumber	Plocation	Dnum
Product X	1	Dhahran	5
Product Y	2	Jubail	5
Product Z	3	Khobar	5
Computerization	10	Dammam	4
Reorganization	20	Khobare	1
Newbenefit	30	Dammam	4

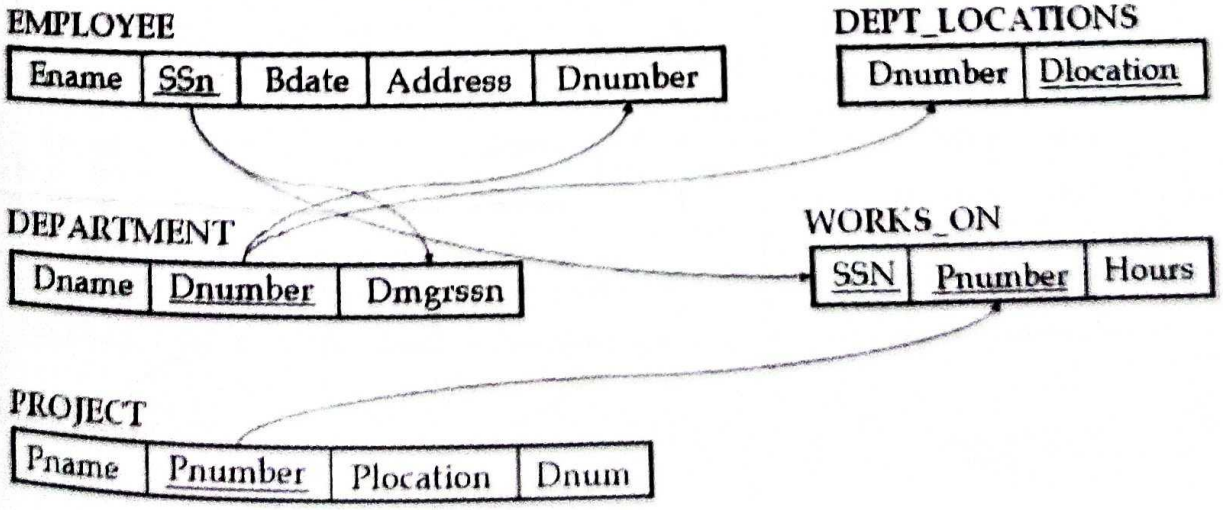
WORKS _ON

Ssn	Pnumber	Hours
111	1	32.5
111	2	7.5

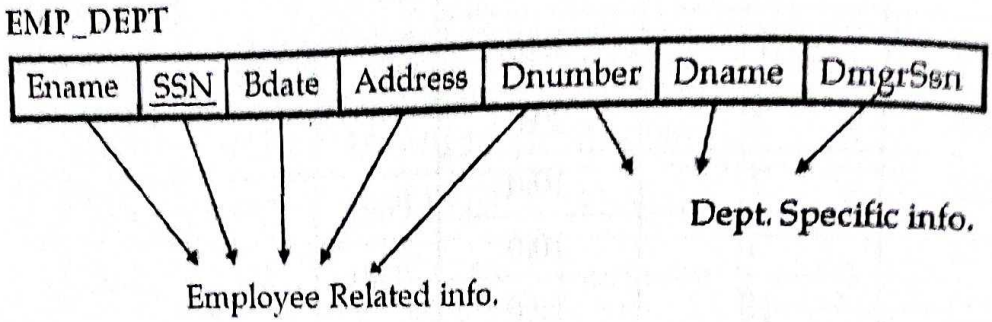
555	3	40.0
666	1	20.0
666	2	20.0
222	2	10.0
222	3	10.0
222	10	10.0
222	20	10.0
333	30	30.0
333	10	10.0
777	10	35.0
777	30	5.0
444	20	15.0
444	20	15.0
888	20	null

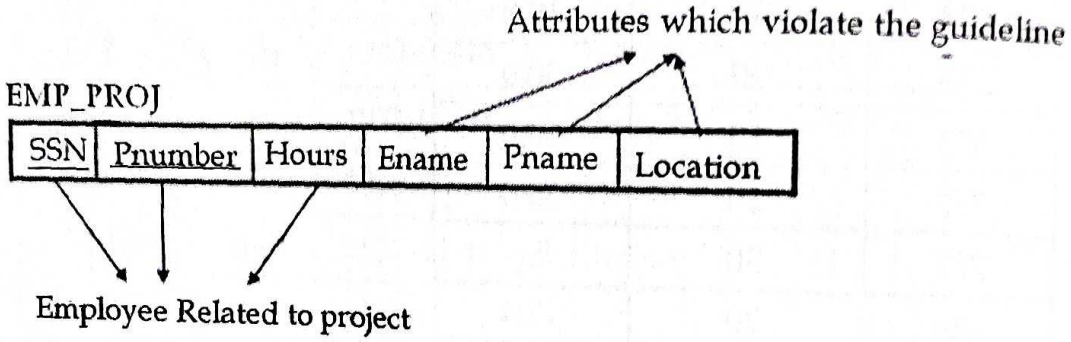
DEPT_LOCATIONS

Dnumber	Plocation
1	Khobar
4	Dammam
5	Dhahran
5	Jubail
5	Khobar



b) Example of Bad Design





4.1.2. Guideline 2: Reducing The Redundant Values in Tuples

- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relation.
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.
- Due to improper grouping of attributes into a relation schema, the following problems are encountered.
  - ❖ Storage wastage
  - ❖ Delete anomalies
  - ❖ Modification anomalies

a) Storage Wastage

Redundant data

Ename	SSN	Ddate	Address	Dnumber	Dname	DMGR Manager
Salah	111	1965-01-09	Khobar	5	Research	222
Wael	222	1955-12-08	Khobar	5	Research	222
Zaher	333	1968-07-19	Dammam	4	Administration	444
Walid	444	1941-06-20	Dhahran	4	Administration	444
Nasser	555	1962-09-15	Jubail	5	Research	222
Essam	666	1972-07-31	Khobar	5	Research	222
Jabber	777	1969-03-29	Khobar	4	Administration	444
Bader	888	1937-11-10	Khobar	1	Headquarters	888

b) Insert Anomalies

EMP_DEPARTMENT

Ename	SSN	Ddate	Address	Dnumber	Dname	DMGR Manager
Salah	111	1965-01-09	Khobar	5	Research	222
Wael	222	1955-12-08	Khobar	5	Research	222
Zaher	333	1968-07-19	Dammam	4	Administration	444
Walid	444	1941-06-20	Dhahran	4	Administration	444
Nasser	555	1962-09-15	Jubail	5	Research	222
Essam	666	1972-07-31	Khobar	5	Research	222
Jabber	777	1969-03-29	Khobar	4	Administration	444
Bader	888	1937-11-10	Khobar	1	Headquarters	888
All	999	1967-10-15	Dammam			
				6	Accounting	111

A null primary key

Newly inserted rows

Delete Anomalies

EMP_DEPARTMENT

Ename	SSN	Ddate	Address	Dnumber	Dname	DMGR Manager
Salah	111	1965-01-09	Khobar	5	Research	222
Wael	222	1955-12-08	Khobar	5	Research	222
Zaher	333	1968-07-19	Dammam	4	Administration	444
Walid	444	1941-06-20	Dhahran	4	Administration	444
Nasser	555	1962-09-15	Jubail	5	Research	222
Essam	666	1972-07-31	Khobar	5	Research	222
Jabber	777	1969-06-29	Khobar	4	Administration	444
Bader	**888**	**1937-11-10**	**khobar**	**1**	**Headquarters**	888

Deleting Borg. James will delete the only information of department 1 from the database

d) Modification Anomalies

EMP_DEPARTMENT

Ename	SSN	Ddate	Address	Dnumber	Dname	DMGR Manager
Salah	111	1965-01-09	Khobar	5	Research	444
Wael	222	1955-12-08	Khobar	5	Research	222
Zaher	333	1968-07-19	Dammam	4	Administration	444
Walid	444	1941-06-20	Dhahran	4	Administration	444
Nasser	555	1962-09-15	Jubail	5	Research	222
Essam	666	1972-07-31	Khobar	5	Research	222
Jabber	777	1969-06-29	Khobar	4	Administration	444
Bader	888	1937-11-10	khobar	1	Headquarters	888

4.1.3. Guideline 3: Reducing Null Values in Tuples

- If possible avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable, make sure they apply in exceptional cases only and not to majority of tuples in a relation.
- Problems with null values :
  - ❖ Waste of disk space
  - ❖ Problem of understanding the meaning of attributes

- Problem in specifying JOIN operations
- Problem in applying some aggregate functions
- May have multiple interpretations (not applicable, unknown, unavailable)

1.4. Guideline 4: Disallowing generating Spurious Tuples

Make sure that the foreign keys refer to unique keys.

Lid	Fname	Lname	Salary
1	Adel	Adam	1000
2	Ageel	Hassan	1100
3	Adel	Khaled	1200

Lectures

SID	Sname	Lname
ICS334	DB	Adel
ICS434	OS	Yahya
ICS202	DS	Adel

Subjects

which Adel is the teacher of DB

Lid	Fname	Lname	Salary
1	Adel	Adam	1000
2	Ageel	Hassan	1100
3	Adel	Khaled	1200

Lectures

SID	Sname	Lname
ICS334	DB	Adel
ICS434	OS	Yahya
ICS202	DS	Adel

Subjects

Now,

SELECT L.Lid, L.Fname, S.Sname FROM lecturers L, Subjects S Where S.Fname = L.Fname

Lid	Fname	Sname
1	Adel	Adam
2	Yahya	OS
3	Adel	DB
1	Adel	DS
2	Adel	DS

- Design relational schemas so that they can be joined with equality conditions of attributes that are easier primary keys or foreign keys in a way that guarantees that no spurious tuples are generated. Do not have relations that contain matching attributes other than foreign key-primary key combination. If such relations are unavoidable, do not join them on such attributes, because the join may produce spurious tuples.

EMP_PROJ1

SSN	Pnumber	Hours	Pname	Plocation

EMP_LOCS

Ename	Plocation

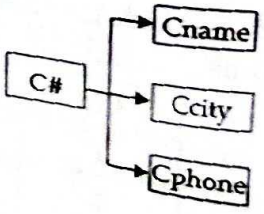
EMP_PROJ

SSN	Pnumber	Hours	Ename	Pname	Plocation

4.2 Define Functional dependencies

- The role of determinants is also expressed as "functional dependencies" whereby we can say, "If an attribute A is a determinant of an attribute B, then B, is said to be functionally dependent on A"
- And likewise "Given a relation R, attribute B of R is functionally dependent on attribute A if and only if each A-value in R has associated with it one B-value in R at any one time".
- "C# is a determinat of Cname, Ccity and Cphone" is thus also "Cname, Ccity and Cphone are functionally dependent on "C# ".
- Given particular value of Cname value, there exists precisely one corresponding value for each of Cname, Ccity and Cphone.
- This is more clearly seen via the following functional dependency diagram:

Transaction						
C#	**Cname**	**Ccity**	**Cphone**	**P#**	**Date**	**Qnt**
1	Swati	Pune	2263035	1	19-12-2000	20
1	Swati	Pune	2263035	2	20-11-2004	30
2	Ruchita	Mumbai	5555910	1	02-04-1994	25
3	Ushma	Pune	223491	2	14-05-2003	20



- Similarly, "(C#, P#, Date) is a determinant of Qnt" is thus also "Qnt is functionally dependent on the set of attributes (C#, P#, Date)". The set of attributes is also known as a composite attribute.

Transaction

C#	Cname	Ccity	Cphone	P#	Date	Qnt
1	Swati	Pune	2263035	1	19-12-2000	20
1	Swati	Pune	2263035	2	20-11-2004	30
2	Ruchita	Mumbai	5555910	1	02-04-1994	25
3	Ushma	Pune	223491	2	14-05-2003	20

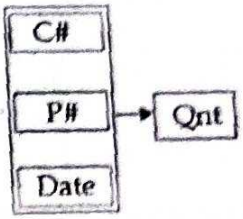
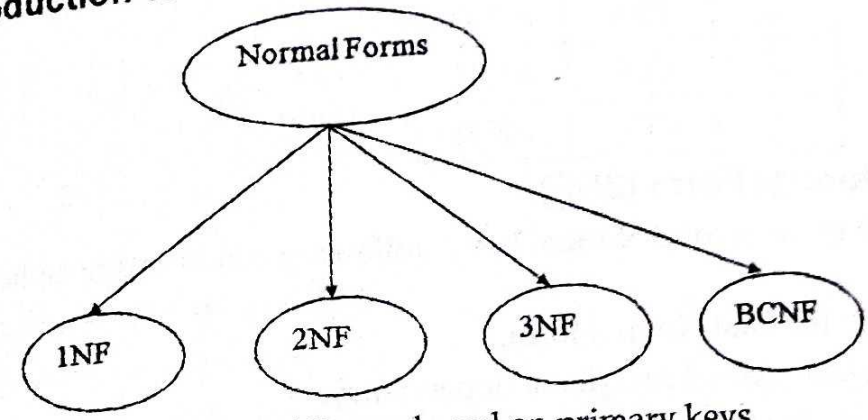


fig. Functional dependency on a composite attribute

4.3 List Normal forms based on primary keys

4.3.0. Introduction to Normal Forms



- The following are the Normal Forms based on primary keys
  - 1NF
  - 2NF
  - 3NF
  - BCNF

4.4 Explain General Definition of first, second and third normal forms, Boyce-codd Normal form with examples

4.4.1. First Normal Form (1NF)

- The table is said to be in 1NF if
  - It should only have single (atomic) valued attributes/columns.
  - Values stored in a column should be of the same domain
  - All the columns in a table should have unique names.
  - And the order in which data is stored, does not matter.
- Consider following Student table.

sid	sname	Phone
1	AAA	11111
		22222
2	BBB	33333
3	CCC	44444
		55555

- As there are multiple values of phone number for sid 1 and 3, the above table is not in 1NF. we can make it in INF. The conversion is as follows

Sid	Sname	Phone
1	AAA	11111
1	AAA	22222
2	BBB	33333
3	CCC	44444
3	CCC	55555

4.4.2. Second Normal Form (2NF)

- For a table to be in the Second Normal Form, following conditions must be followed
  - ❖ It should be in the First Normal form.
  - ❖ It should not have partial functional dependency.
- For example, Consider following table in which every information about the student is maintained in a table such as student id(sid), student name (sname), course id (cid) and course name (cname).

STUDENT_COURSE

sid	sname	cid	cname
1	AAA	101	C
2	BBB	102	C++
3	CCC	101	C
4	DDD	103	java

- This table is not in 2NF. For converting above table to 2NF we must follow the following steps:
  - ❖ **Step 1:** The above table is in 1NF.
  - ❖ **Step 2:** Here sname and sid are associated similarly cid and cname are associated with each other. Now if we delete a record with sid = 2, then automatically the course C++ will also get deleted. Thus,

Sid->sname or cid->cname is a partial functional dependency, because {sid, cid} should be essentially a candidate key for above table. Hence to bring the above table to 2NF we must decompose it as follows:

Sid	Sname	cid
1	AAA	101
2	BBB	102
3	CCC	101
4	DDD	103

→ Here candidate key is (sid, cid) and (sid, cid)→Sname

cid	cname
101	C
102	C++
101	C
103	Java

→ Here candidate key is cid
Here cid→cname

Thus now table is in 2NF as there is no partial functional dependency

4.4.3. Third Normal Form(3NF)

- A table is said to be in the Third Normal Form when,
  - ❖ It is in the Second Normal form .(i.e. it does not have partial functional dependency)
  - ❖ It doesn't have transitive dependency.
- For example, Consider following table Student_ details as follows

Sid	sname	zipcode	cityname	state
1	AAA	11111	Pune	Maharashtra
2	BBB	22222	Surat	Gujarat
3	CCC	33333	Chennai	Tamilnadu
4	DDD	44444	Jaipur	Rajastan
5	EEE	55555	Mumbai	Maharashtra

- Here
  - ❖ **Super keys:** {sid}, {sid,sname},{sid,sname,zipcode}, {sid, zipcode, cityname}.....and so on.
  - ❖ **Candidate Keys:**{sid}
  - ❖ **Non-Prime Attributes:** {sname, zipcode, cityname, state}

- The dependencies can be denoted as
  - sid→sname
  - sid→zipcode
  - zipcode→cityname
  - cityname→state

The above denotes the transitive dependency. Hence above table is not in 3NF. We can convert it into 3NF as follows:

Student

sid	sname	zipcode
1	AAA	11111
2	BBB	22222
3	CCC	33333
4	DDD	44444
5	EEE	55555

Zip

zipcode	cityname	state
11111	Pune	Maharashtra
22222	Surat	Gujarat
33333	Chennai	Tamilnadu
44444	Jaipur	Rajasthan
55555	Mumbai	Maharashtra

4.4.4. Boyce-codd Normal form (BCNF)

- Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handle by 3NF.
- A 3NF table which does not have multiple Overlapping candidate keys is said to be in BCNF.
- In other words, For a table to be in BCNF, following conditions must be satisfied:
  - R must be in 3rd Normal Form
  - For each functional dependency (X→Y), X should be a super Key. In simple words if Y is a prime attribute then X can not be non prime attribute.
- For example, Consider following table that represents that a Student enrolment for the course

Enrollment Table

Sid	Course	Teacher
1	C	Ankita
1	Java	Poonam
2	C	Ankita
3	C++	Supriya
4	C	Archana

- From above table following observations can be made
  - One student can enroll for multiple courses. For example student with sid = 1 can enrol for C as well as Java.
  - For each course, a teacher is assigned to the student.
  - There can be multiple teachers teaching one course for example course C can be taught by both the teachers namely-Ankita and Archana.
  - The candidate key for above table can be (sid, course), because using these two columns we can find
  - The above table holds following dependencies
    - (sid, course)→Teacher
    - Teacher→course
  - The above table is not in BCNF of the dependency teacher→course. Note that the teacher is not a superkey on in other words, teacher is a non prime attribute and course is a prime attribute and non-prime attribute derives the prime attribute.
  - To convert the above table to BCNF we must decompose above table into Student and Course tables

Student

Sid	Teacher
1	Ankita
1	Poonam
2	Ankita
3	Supriya
4	Archana

Course

Teacher	Course
Ankit	C
Poonam	Java
Ankit	C
Supriya	C++
Archana	C

- Now the table is in BCNF

4.5 Define Transaction in DBMS

- A transaction can be defined as a group of tasks that form a single logical unit.
- For example, Suppose we want to withdraw Rs. 100 from an account then we will follow following operations :
  - Check account balance
  - If sufficient balance is present request for withdrawal.
  - Get the money
  - Calculate Balance = Balance -100
  - Update account with new balance.
- The above mentioned four steps denote one transaction
- In database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

4.6 Illustrate the ACID Properties of Transactions

4.7 Describe about desirable Properties of Transaction

4.6.1. ACID Properties of Transactions (or) Desirable Properties of Transaction

a) Atomicity

- This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.
- No transaction in the database is left half completed.
- Data should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.

- For example, In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e. customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

b) Consistency

- The database must remain in consistent state after performing any transaction.
- For example, In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

c) Isolation

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transaction will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- For example, If a bank manger is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

d) Durability

- The database should be strong enough to handle any system failure.
- If there is any set of inset/update, then it should be able to handle and commit to the database.
- If there is any failure, the database should be able to recover it to the consistent state.
- For example, In ATM withdrawal example, if the system failure happens after Customer getting the money then the system should be strong enough to update Database with his new balance, after system recovers. For that purpose the system has to keep the log of each transaction and its failure. So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to Database.

.8 Illustrate Commit, Rollback, and Save Point

.8.0. Introduction

- Transaction control language (TCL) commands are used to manage transactions in the database. These are used to manage transactions in the database.
- These are used to manage the changes made to the data in a table by DML statements.
- It also allows statements to be grouped together into logical transactions.

4.8.1. COMMIT Command

- Commit commands is used to permanently save any transaction into the database.
- When we use any DML command like INSERT, UPDATE OR DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
- To avoid that, we use the COMMIT command to mark the changes as permanent.
- **Syntax:**

```
COMMIT;
```

4.8.2. ROLLBACK Command

- This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.
- If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.
- **Syntax:**

```
ROLLBACK TO savepoint_name;
```

4.8.3. SAVEPOINT Command

- SAVEPOINT command is used to save a transaction temporarily so that you can rollback to that point whenever required.
- **Syntax:**

```
SAVEPOINT savepoint_name;
```

- Using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

8.4. Using Savepoint and Rollback

- Following is the table CLASS

id	name
1	Abhi
2	Adam
4	Alex

- Following are the few SQL queries on the above table and see the results.

```
INSERT INTO class VALUES (5, 'Rahul');
COMMIT;
UPDATE class SET name = 'Abhijit' WHERE id = '5';
SAVEPOINT A;
INSERT INTO class VALUES (6, 'chris');
```

```
SAVEPOINT B;
INSERT INTO class VALUES (7, 'Bravo');
SAVEPOINT C;
SELECT * FROM class;
```

Note: SELECT statement is used to show the data stored in the table.

- The resultant table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Ahbijit
6	Chris
7	Bravo

- Use the ROLLBACK command to roll back the state of data to the **savepoint B**

```
ROLLBACK TO B;
SELECT * FROM class;
```

- Resultant class table will look like

id	name
1	Abhi
2	Adam
4	Alex
5	Ahbijit
6	Chris

- Now Again use the ROLLBACK command to roll back the state of data to the savepoint A

```
ROLLBACK TO A;
SELECT * FROM class;
```

- Now the Resultant table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Ahbijit

4.8.5. Differences between ROLLBACK and COMMIT

S.No.	ROLLBACK	COMMIT
1	ROLLBACK command is used to undo the changes made by the DML commands.	The COMMIT command is used to save the modifications done to the database values by the DML commands.
2.	It rollbacks all the changes of the current transaction.	It will make all the changes permanent that cannot be rolled back.
3.	**Syntax:** DELETE FROM table_name ROLLBACK	**Syntax:** COMMIT;

4.9 Explain Serializability

4.9.1. Serializability

- When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transactions).
- Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.
- For example,

T_1	A	B	T_2
Initial Value	100	100	
A = A - 10			
W(A)			
B = B + 10			
W(B)			
	90	110	
			A = A - 10
			W(A)
	80	110	

- ❖ In above transactions initially T_1 will read the values from database as A = 100, B=100 and modify the values of A and B. But transaction T_2 will read the modified value i.e. 90 and will modify it to 80 and perform write operation.
- ❖ Thus at the end of transaction T_1 value of A will be 90 but at end of transaction T_2 value of A will be 80.
- ❖ Thus conflicts or inconsistency occurs here.This sequence can be converted to a sequence which may give us consistent result. This process is called Serializability.

- There are two types of Serializabilites : Conflict Serializability and View Serializability.

4.9.2. Conflict Serializability

- Suppose T_1 and T_2 are two transactions and I_1 and I_2 are the instructions in T_1 and T_2 respectively. Then these two transactions are said to be conflict Serializable, if both the instruction access the data item d, and at least one of the instruction is write operation.

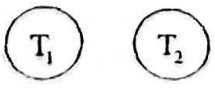
- In the definition three conditions are specified for conflict serializability
  - There should be different transactions
  - The operations must be performed on same data items
  - One of the operation must be the Write(w) operation
- For Example, Consider the following the two transactions and schedule (time goes from top to bottom)

T_1	T_2
R(A)	
W(A)	R(A)
	R(B)
R(B)	
W(B)	

  - **Step 1:** To check whether the schedule is conflict serializable or not we will check from top to bottom. Thus we will start reading from top to bottom as

    $T_1 : R(A) \rightarrow T_1 : W(A) \rightarrow T_2 : R(B) \rightarrow T_2 : R(B) \rightarrow T_1 : R(B) \rightarrow T_1 : W(B)$

  - **Step 2:** Two operations are called as conflicting operations if all the following conditions hold true for them
    - Both the operations belong to different transactions.
    - Both the operations are on same data item.
    - At least one of the two operations is a write operation
  - From above given example in the top to bottom scanning we find the conflict as $T_1 : W(A) \rightarrow T_2 : R(A)$.
    - Here note that there are two different transactions T_1 and T_2
    - Both work on same data item i.e. A
    - One of the operation is write operation
  - **Step 3:** We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions, there will be two nodes namely T_1 and T_2



- **Step 4:** Draw the edge between conflicting transactions. For example, In above given scenario, the conflict occurs while moving from $T_1 : W(A)$ to $T_2 : R(A)$. Hence edge must be from T_1 and T_2.

- **Step 5:** Repeat the step 4 while reading from top to bottom. Finally the precedence graph will be as follows

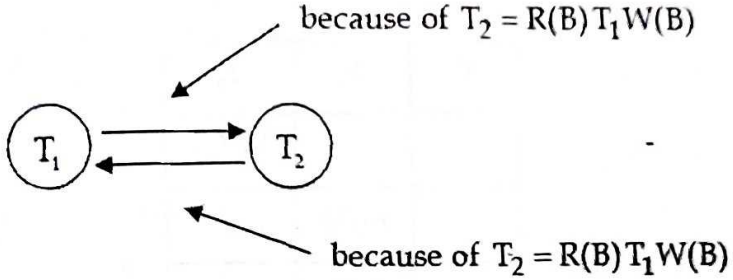


fig. Precedence Graph

- **Step 6:** Check if any cycle exists in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the cycle is found then schedule is not conflict Serializable. In the step 5, We get a graph with cycle, that means given schedule is not conflict Serializable.

4.9.3. View Serializability

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a View Serializable Schedule.
- **View Equivalent Schedule:** Consider two schedules S_1 and S_2 consisting of transactions T_1 and T_2 respectively, then schedules S_1 and S_2 are said to be view equivalent schedule if it satisfies following three conditions:
  - If transaction T_1 reads a data item A from the database initially in schedule S_1, then in schedule S_2 also, T_1 must perform the initial read of the data item X from the database. This is same for all the data items. In other words, the initial reads must be same for all data items.
  - If data item A has been updated at last by transaction T_1 in schedule S_1, then in schedule S_2 also, the data item A must be updated at last by transaction T_1
  - If Transaction T_1 reads a data item that has been updated by the transaction T_1 in schedule S_1, then in schedule S_2 also, transaction T_1 must read the same data item that has been updated by transaction T_1. In other words the Write-Read sequence must be same.

Steps to check whether the given schedule is View Serializable or not

- **Step 1:** If the schedule is conflict serializable then it is surely view serializable because conflict serializability is a restricted form of view serializability.
- **Step 2:** If it is not conflict serializable schedule then check whether there exist any blind write operation. The blind write operation is a write operation without reading a value. If there does not exist any blind write then that means the given schedule is not view serializable. In other words if a blind write exists then that means schedule may or may not be view conflict.
- **Step3:** Find the view equivalence schedule.

- For example,

T_1	T_2	T_3
		W(C)
	R(A)	
	W(B)	
R(C)		
		W(B)
W(B)		

- The initial read operation is performed by T_2 on data item A or by T_2 on data item C. Hence we will begin with T_2 to T_2. We will choose T_2 at the beginning.
- The final write is performed by T_2 on the same data item B. Hence T_1 will be at the last position.
- The data item C is written by T_2 and then it is read by T_1 Hence T_1 should appear before T_2.
- Thus we get the order of schedule of view serializability as $T_1 - T_2 - T_2$

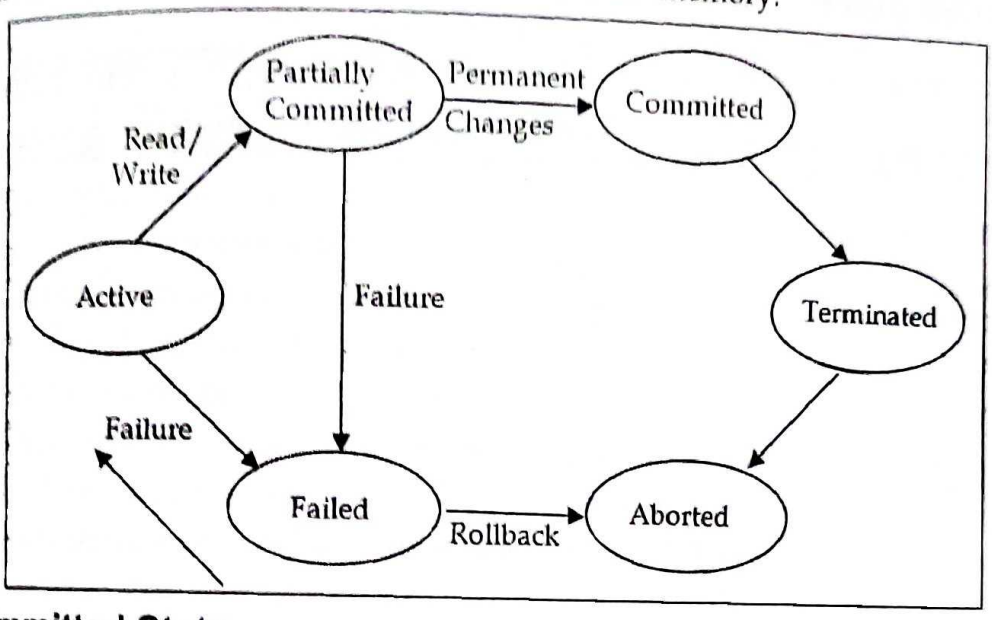
4.10. Give the states of Transactions

1. Active State

- This is the first state in the life cycle of a transaction.
- A transaction is called in an active state as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

2. Partially Committed State

- After the last instruction of transaction has executed, it enters into a partially committed state.
- After entering this state, the transaction is considered to be partially committed.
- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.



3. Committed State

- After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
- Now, the transaction is considered to be fully committed.

Note:

- ❖ After a transaction has entered the committed state, it is not possible to roll back the transaction.
- ❖ In other words, it is not possible to undo the changes that has been made by the transaction.
- ❖ This is because the system is updated into a new consistent state.
- ❖ The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operations.

4. Failed State

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it become impossible to continue the execution, it enters into a failed state.

5. Aborted State

- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state.

6. Terminated State

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.