RDBMS (ACCS – 16405)

Subject Notes

RELATIONAL ALGEBRA

Relational Algebra is a collection of operations used to manipulate relations (tables). These operations enable the users to specify the retrieval requests which results in a new relation built from one or more relations.

Relational Algebra is a procedural language, which specifies, the operations to be performed on the existing relations to derive result relations. It is a procedural language which means that user has to specify what is required and what is the sequence of steps performed on the database to obtain the required output.

It is the important that the result of use of relational algebraic operations on Relations must themselves be a relation. This is because operators can be used sequentially in various combinations to obtain desired results.

The relational algebraic operations can be divided into 2 groups:

- **Basic Set Oriented Operations** or **Traditional Set operations** derived from Mathematical Set theory. They are applicable because each relation is defined to be set of Tuples. These include union, intersection, difference, Cartesian product.

- **Special Relational Operations** These include join, selection, projection and division. These operations were designed specifically for relational databases. These operations don‟t add only power to the algebra but simply for common queries that are lengthy to express using basic set oriented operations.

BASIC SET ORIENTED OPERATIONS

The fundamental or basic set oriented operations consists of Union (U), Intersection ($\cap$), Difference (-) and Cartesian Product (×). All of these operations are binary operations which means that operation is applied on pair of operands i.e. relations. Three of these four basic operations-UNION , INTERSECTION and DIFFERENCE require that operand relations must be union compatible.

Two relations are union compatible if the following conditions are satisfied

- Both the relations must have the same number of attributes i.e. they should have same degree.

- Each column of the first relation must have the same data type as the corresponding column of the second relation. There are no requirements for the relevant attributes names to be the same.

Id	Name
501	Amity
503	Anand
504	Amitabh
507	Rajinder
510	Kapil
512	Ranjit

Id	Name
503	Anand
504	Amitabh
506	Madhav
510	Kapil

The above 2 relations are union compatible as they have same number of attributes and are drawn from common domain. Although attributes names are different it hardly makes any difference.

1. <u>UNION</u>

The Union operations or 2 relations results in a new table containing rows from both tables with duplicated removed. The 2 relations used in the union operation must be union compatible i.e. they must have the same number of columns and corresponding columns in the select statement must come from the same domain.

P

Id	Name
501	Amit
503	Anand
504	Amitabh
507	Rajinder
510	Kapil
512	Ranjit

Q

Id	Name
503	Anand
504	Amitabh
506	Madhav
510	Kapil

R=PUQ

Id	Name
501	Amit
503	Anand
504	Amitabh
506	Madhav
507	Rajinder
510	Kapil
512	Ranjit

Properties of Union operation:

1. Commutativity- It means that result of (PUQ) is same as (QUP)
2. Associativity- it means that PU(QUS)=(PUQ)US where P,Q and S are relations.

2. INTERSECTION

The intersection of 2 relations results in a new table containing rows that are common to both tables. The two relation used in Intersection operation must be union compatible. The intersection operation is a binary operation and the notation for the intersection of two relations P and Q is ∩ .

P

Id	**Name**
501	Amit
503	Anand
504	Amitabh
507	Rajinder
510	Kapil
512	Ranjit

Q

Id	**Name**
503	Anand
504	Amitabh
506	Madhav
510	Kapil

R= $P \cap Q$

Id	**Name**
503	Anand
504	Amitabh
510	Kapil

The result R obtained by taking intersection of P and Q contains all the rows which are common to both the table.

Properties of Intersection operation:

1. The input relations must be union compatible.
2. Commutativity- It means that result of $P \cap Q$ is same as that Q∩P.
3. Associativity- It means that $P \cap (Q \cap S) = (P \cap Q) \cap S$ where P,Q and S are relations.

3. DIFFERENCE

The difference of two relation results in a new relation that contains those tuple that occur in the first relation but not in the second relation. The two relations in the difference operation must be union compatible.

P

Id	Name
501	Amit
503	Anand
504	Amitabh
507	Rajinder
510	Kapil
512	Ranjit

Q

Id	Name
503	Anand
504	Amitabh
506	Madhav
510	Kapil

R=P-Q

Id	Name
501	Amit
507	Rajinder
512	Ranjit

The result R obtained by taking difference of 2 relations P & Q, where P is the first relation and Q is the second relation, contains all the rows that are contained in relation P but not in Relation Q.

Properties of Difference Operations:

1. The input relations must be union compatible.
2. They are not commutative – It means that the result of P-Q is not the same as the result of Q-P.

   P-Q≠Q-P.
3. They are not associative- It means that P-(Q-S)≠(P-Q)-S where P,Q,S are relations

4. CARTESIAN PRODUCT (×)

The Cartesian product of 2 relations results in a new relation that includes concatenation of every tuples of first relation with every tuple of second relation. In other words the new relation is created consisting of all possible combination of the tuples. It is also known as Cross product or cross join. It is not necessary for the relation on which this operation is to be performed to be union compatible. The Cartesian product is a binary operation which means that it always operates on 2 relations.

The notation for the Cartesian product of Relations P & Q is P × Q

P

Id	**Name**
501	Amit
503	Anand
504	Amitabh

Q

Id	**Name**
503	Anand
506	Madhav

R=P×Q

P.Id	**Name**	**Q.Id**	**Name**
501	Amit	503	Anand
501	Amit	506	Madhav
503	Anand	503	Anand
503	Anand	506	Madhav
504	Amitabh	503	Anand
504	Amitabh	506	Madhav

Properties of Cartesian product operation:

1. The relations on which Cartesian product operation is applied need not necessary to be union compatible.
2. The resultant relation may hold duplicate attributes if some attributes of 2 relations are defined on common domains.
3. The degree of the resultant operation is equal to sum of the degree of all the relations i.e. Degree of R= Degree of P + Degree of Q
4. The total number of rows in the resultant operation is equal to the product of the number of the first and the second relation i.e.
   Total number of tuples of R= Total number of tuples of P × Total number of tuples of Q.

RELATIONAL OPERATIONS

1. SELECTION (σ)

The selection operation also known as restriction operation results in a new relation that contains only those rows of relations that satisfy a specified condition. It is a unary operation which means that it can work only on a single relation. The selection operation is denoted by lowercase Greek sigma (σ) with a predicate appearing as a subscript and relation name is given in parenthesis following the sigma

i.e. $\sigma_{<Condition>}(R)$

By the term predicate we mean to say, the property that the individual object posseses. It can be analogusly used with the term condition.

Suppose an Employee relation is

Employee

Id	**Name**	**Salary**
501	Amit	10000
503	Anand	6000
504	Amit	10000
507	Rajinder	8000
509	Anand	9000
510	Kapil	12000
512	Ranjit	15000

Result of Selection

Id	**Name**	**Salary**
501	Amit	10000
504	Rajinder	8000
510	Kapil	15000
512	Ranjit	12000

$\sigma_{Salary \geq 8000}$(Employee)

Now we have to find all employees having salary ≥ 8000 in the relation. The selection operation is denoted as $\sigma_{Salary \geq 8000}$(Employee). Thus, the result or restriction which is applied on EMPLOYEE relation satisfying the condition salary ≥ 8000 is shown. It contains all the tuples from the relation where salary of the employee is greater than or equal to 8000.

The selection operation can also use other comparison operators such as greater than(>), less than(<), less than equal to (≤), not equal to(≠) and greater than equal to (≥) and the logical operators such as AN,OR,NOT.

PROPERTIES OF SELECTON OPERATION

1. It is an unary operation i.e. it can operate only on a single relation.
2. The resulting relation has the same degree as that of the original relation. This means that the number of columns in both the relations is same.
3. The number of rows of the resulting relation is always less than or equal to number of rows of the original relation
4. Commutativity- The selection operation is commutative which mean that $\sigma_{<cond\ 1>}(\sigma_{<cond\ 2>}(R)) = \sigma_{<cond\ 2>}(\sigma_{<cond\ 1>}(R))$

PROJECTION (∏)

The project operation results in a new relation that contains a subset of columns of a relation and eliminates any duplicate rows that may result. It is a unary operation which means that it operates only on a single relation.

The projection operation is denoted by a Greek capital letter Pi(∏) and the attribute to be retrieved appear as subscripts separated by commas and relation name in given is parenthesis following the P_i i.e. $\prod_{<attribute\ 1>,<attribute\ 2>,\ldots\ldots\ldots<attribute\ n>}(R)$

Suppose an EMPOLYEE relation having the attributes, Id, name and salary where Id is the primary key, suppose that we want a list of name of employees in the Employee table in Fig A.

Also suppose that we want a list of name and salary details of each employee in employee table shown in fig. B

Employee

Id	**Name**	**Salary**
501	Amit	10000
503	Anand	6000
504	Amit	10000
507	Rajinder	8000
509	Anand	9000
510	Kapil	12000
512	Ranjit	15000

Result of Projection 1

Name
Amit
Anand
Rajinder
Kapil
Ranjit

$\prod_{Name}$(Employee)

Fig. A

Result of Projection 2

Name	**Salary**
Amit	10000
Anand	6000
Rajinder	8000
Anand	9000
Kapil	12000
Ranjit	15000

$\prod_{Name,\ Salary}$(Employee)

Fig. B

Properties of Projection Operation

1. It is an unary operation i.e. it operates only on a single relation.
2. The degree of the resulting relation is equal to number of attributes specified in the attribute list
3. If the attribute list contains a primary key attribute then the number of tuples in the resulting relation is equal to the number of tuples in the original relation.
4. Non Commutative- It doesn‟t hold the commutative property.

JOIN (⋈)

The join operation results in a new relation that consists of tuples resulting from combining the related rows from 2 relations. It forms a new relation which contains all the attributes from both the joined relations whose tuples are those defined by the restriction applied i.e. the join condition applied on 2 participating relation. The join is performed on two relations, which have one or more attributes in common. These attributes must be domain compatible i.e. they have the same data type.

Since the join operation is performed on two relations so it is a binary operation. The join operation is denoted by a join symbol ( ). The $\bowtie$general form of representing a join operation on 2 relations P & Q is

$$P \bowtie_{\text{<Join condition>}} Q$$

Where join condition is of the form **$p_i \Theta q_i$** wher p_i is an attribute of relation P and q_i is an attribute or relation Q. Both p_i and q_i should be domain compatible i.e. should have same data type. The Θ is one of the comparison operators such as equal to (=), not equal to ($\neq$), less than (<), less than equal to ($\leq$), greater than (>), greater than equal to ($\geq$). A join operation with such a general join condition is called a THETA JOIN.

When we use equality operator, equal to (=) in the join condition (i.e. $P_i=Q_i$) then such a JOIN is called an **EQUI JOIN**. It is one of the most powerful and commonly used join.

If the comparison operators is other than the equal to sign (>,≥,<,≤,≠0 then the JOIN is called **NON-EQUI JOIN** and is very rarely used. Consider the EMP and DEPT table as shown.

EMP

Emp_Id	**Ename**	**Salary**	**Dept_Id**
101	Anurag	12000	01
103	Anshu	10000	01
106	Ankur	15000	02

Dept

Dep_no	**Dname**
01	Finance
02	Marketing
03	Packing

Suppose that we want to know the employee information along with Dname in which each employee is working.

The resultant relation

Emp_Id	**Ename**	**Salary**	**Dept_Id**	**Dep_no**	**Dname**
101	Anurag	12000	01	01	Finance
103	Anshu	10000	01	02	Marketing
106	Ankur	15000	02	03	Packing

$$\textbf{EMP} \bowtie_{\textbf{Dept\_Id=Dep\_No}} \textbf{(DEPT)}$$

The common attributes in both the tables i.e. Dept_Id in EMP table and Dep_No in DEPT table should not have the same names so as to preserve the uniqueness of the column names in the resultant relation (R). if they have the same names then they should be renamed to preserve uniqueness.

Since the EQUI JOIN produce a result containing two identical columns (i.e. Dept_Id and Dep_no in the resultant relation). If one of this column is eliminated then that JOIN is called the NATURAL JOIN which is shown below.

Emp_Id	**Ename**	**Salary**	**Dept_Id**	**Dname**
101	Anurag	12000	01	Finance
103	Anshu	10000	01	Marketing
106	Ankur	15000	02	Packing

Natural JOIN on EMP, DEPT table.

The natural join may also been referred to as INNER JOIN

Another type of JOIN in which a relation is joined to itself by comparing values with a column of a single relation is known as SELF JOIN

EMP Table

Emp_Id	**Ename**	**Mang_Id**
101	Anurag	-
103	Anshu	101
104	Amar	106
106	Ankur	-
107	Puneet	101

The resultant relation

EName	**Mname**
Anurag	Anshu
Amar	Ankur
Puneet	Anshu

Suppose we want find the manger‟s name for all the employees. This would be done using self join because required information is contained in same table. Since self join is also type of join and join is binary operation so it must act on two relations. But since we are having only one relation so we have to make copy or two tables with different names so that can perform join operation with itself.

EMP$\bowtie_{\text{EMP.Mang_Id=MANAGER.Mang_Id}}$(MANAGER)

DIFFERENCE BETWEEN CARTESIAN PRODUCT AND JOIN

The Cartesian product and JOIN seems to be similar but they actually different. The main difference in JOIN operation is that only combination or rows satisfy the join condition appears inthe resultant relation but in Cartesian product all possible combination of rows appears in the resultant relation. So if we don‟t apply any "Join Condition" on the join then it is same as that of Cartesian product. To explains this difference consider the table.

EMP

Emp_Id	**Ename**	**Dept_Id**
101	Anurag	01
103	Anshu	01
106	Ankur	02

DEPT

Dep_no	**Dname**
01	Finance
02	Marketing
03	Packing

EMP$\bowtie_{\text{Dept_Id=Dept_No}}$ (DEPT)

Emp_Id	**Ename**	**Dept_Id**	**Dname**
101	Anurag	01	Finance
103	Anshu	01	Marketing
106	Ankur	02	Packing

EMP×DEPT

Emp_Id	**Ename**	**Dept_Id**	**Dep_no**	**Dname**
101	Anurag	01	01	Finance
101	Anurag	01	02	Marketing
101	Anurag	01	03	Packing
103	Anshu	01	01	Finance
103	Anshu	01	02	Marketing
103	Anshu	01	03	Packing
106	Ankur	02	01	Finance
106	Ankur	02	02	Marketing
106	Ankur	02	03	Packing

Properties of Join Operation

1. Associtivity- Join operation is associative.
2. Rows whose join attributes are null don‟t appear in the resultants relation table
3. More than two tables can also be joined but it increases the complexity.

OUTER JOIN

Some times we may want both the matching as well as non matching tuples returned for the relations that are joined. This type of operation is known as OUTER JOIN. In OUTER JOIN, the rows in one relation having no matching rows in the other relation will also appear in the resultant table with nulls in the other attributes positions instead of being ignored as in case of INNER JOIN. The OUTER JOIN can have one of three types.

- Left OUTER JOIN
- Right OUTER JOIN
- Full OUTER JOIN

The left outer join operation retains all the rows in the first relation and if no matching row is found in the second relation then the attributes of second relation in the resultant relation are filled with null values. It is denoted by ⟕ . The notation for left outer join is P⟕ Q where P is the first relation and Q is the second relation. Consider the EMP and the DEPT table.

EMP

Emp_Id	Ename	Salary	Dept_Id
101	Anurag	12000	01
103	Anshu	10000	01
106	Ankur	15000	02
107	Manav	8000	NULL

DEPT

Dep_no	Dname
01	Finance
02	Marketing
03	Packing

The result of left outer join is performed on EMP as the first relation and the DEPT as the second relation with common field equated, Dept_Id=Dep_no is stored in TEMP. Then projection of EMP_Id, Ename,Salary,Dname is made from temporary relation variable (TEMP) so the result is

Emp_Id	Ename	Salary	Dname
101	Anurag	12000	Finance
103	Anshu	10000	Marketing
106	Ankur	15000	Packing
107	Manav	8000	NULL

Here we see that we have „NULL‟ value in the name field of employee whose EMP_Id=‟107‟. This is because this employee has not yet been allotted a department so its „Dept_Id=Null‟
The righter outer join operation retain all the rows in the second relation and if no matching row is found in the first relation then the attributes of the first relation in the resultant relation are filled with null values.
It is denoted by ⟖ symbol. The notation for right outer join is P ⟖ Q where P is the first relation and Q is the second relation.
Consider the same example as in case of left outer join. Here when the right outer join is performed on the EMP and DEPT relation and then projection of the EMP_Id,Ename, Salary and Dname from the outer join is taken then the result would be

Emp_Id	**Ename**	**Salary**	**Dname**
101	Anurag	12000	Finance
103	Anshu	10000	Marketing
106	Ankur	15000	Packing
Null	Null	Null	Packing

Here we see that all records corresponding to DEPT table are shown in the result. Here matching values result some values, but since corresponding to Dep_no=03 no record is present in EMP table so it results a null values corresponding to the attribute which are from EMP table when join is being performed.

The full outer join operation retains all the rows in both the first and second relations. When no matching rows are found then it fills them with null values as needed.

It is denoted by ⟗ symbol. The notation for full outer join is P ⟗ Q where P is the first relation and Q is the second relation.

NORMALIZATION

Normalization is a designing technique that is widely used as a guide in designing Relational databases. It is a process of decomposing (splitting) the Relations into Relations with fewer attributes by minimizing the redundancy of data and minimizing insertion, deletion and updation anomalies. The Relations with fewer attributes possess all desirable properties. Normalization may be defined as a step by step Reversible process of transforming an unnormalized Relation into Relations with progressively simpler structures. Since process is reversible no information is lost in transformation.

We normalize the Relational Database Management System because of the following reasons:

- Minimize data redundancy.
- To make Database structure flexible.
- Data should be consistent throughout the database i.e. it should not suffer from following anomalies.

Insert Anomaly- Due to lack of data i.e. all data available for Insertion such that null values in keys should be avoided. This kind of Anomaly can seriously damage a database.

Update Anomaly- It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.

Deletion Anomaly- It leads to loss of data for rows that are not stored elsewhere. It could result in loss of vital data.

- Complex queries required by the user should be easy to handle.
- On decomposition of a Relation into smaller Relations with fewer attributes on normalization, the resulting relations whenever joined must result in the same Relation without any extra rows. The join operation can be performed in any order. This is known as Lossless Join decomposition.
- The Resulting Relations (tables) obtained on normalization should possess the properties such as each row must be identified by a unique key, no repeating groups, homogenous columns, each column is assigned a unique name etc.

ADVANTAGES OF NORMALIZATION

The following are the advantages of the Normalization.

- More efficient data structure.
- Avoid redundant fields or columns.
- More flexible data structure i.e. we should be able to add new rows and data values easily.
- Ensures that distinct tables exist when necessary.
- Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled.
- Minimizes data duplication.

- Close Modeling of real world entities, processes and their relationship.

DISADVANTAGES OF NORMALIZATION

The following are the disadvantages of Normalization.

- You cannot start building the database before you know what the user needs.
- On normalizing the Relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
- It is very time consuming and difficult process in normalizing Relations of higher degree.
- Careless decomposition may lead to bad design of database which may lead to serious problems.

FIRST NORMAL FORM (1NF)

A Relation is said to be in First Normal Form (1NF) if and only if it follow the rules.

- All the Primary key attributes are defined.
- There are no repeating groups in the table.
- All attributes are dependent on the primary key.

A relation to be in 1NF, a domain of an attribute must contains atomic values and the value an attribute contains in a row must be a single value from a domain of that attribute. To explain the first normal form, consider the example of STUDENT table which contains the past performance of students of a particular class.

STUDENT (St_Id, St_Name, RECORD(Subject, Marks)). Thus the STUDENT table contains the student‟s Roll number (St_Id), the name of the student (St_Name), the subject he studies (Subject_Rec) and marks obtained in each subject (Marks_Rec).

St_Id	St_Name	RECORD	
		Subject	Marks
2407	Ranjit	Computer	72
		Economics	65
		Maths	79
2408	Anurag	English	63
		Punjabi	75
		Economics	89

STUDENT TABLE

The above STUDENT table is an unnormalized table as it contains multiple values corresponding to **Subject** and **Marks** attributes i.e. these values are non-atomic. So tables with multi value entries are called unnormalized tables.

To overcome these problems we have to eliminate the non atomic values of subject and marks attributes.

St_Id	St_Name	Subject_Rec	Marks_Rec
2407	Ranjit	Computer	72
2407	Ranjit	Economics	65
2407	Ranjit	Maths	79
2408	Anurag	English	63
2408	Anurag	Punjabi	75
2408	Anurag	Economics	89

STUDENT_1 TABLE

There are two ways to achieve the First Normal Form.

Method 1: To remove the repeating values for a column the STUDENT table was converted to a flat table STUDENT_1 by repeating the pair (St_Id, St_Name) for every entry in the table and consequently removing the attribute RECORD with its subordinates attributes Subject_Rec and Marks_Rec.

Method 2: Another method to convert the unnormalized table into 1NF is by decomposition of the original relation, in which the Student table was decomposed into two sub-tables STU_DETAILS and STU_PERFORMANCE.

STU_DETAILS

St_Id	St_Name
2407	Ranjit
2408	Anurag

STU_PERFORMANCE

St_Id	Subject	Marks
2407	Computer	72
2407	Economics	65
2407	Maths	79
2408	English	63
2408	Punjabi	75
2408	Economics	89

The Relation (table) are decomposed according to the following Rules –

- One of the Relation (table) consists of the Primary Key (i.e. St_Id) of the original table (i.e. STUDENT) and non repeating attributes of the original table (i.e. St_Name).
- The other Relation Consists of copy of the Primary key (i.e. St_Id) of the original table (i.e. STUDENT) and all the repeating attributes of the Original table (i.e. Subject, Marks)

FUNCTIONAL DEPENDENCY

Functional Dependency is the basis for First three normal forms. The functional dependencies are the consequence of the interrelationships among attributes of a Relation (table) represented by some link or association.

An Attribute (column) Y of a Relation (table) R is said to be functionally dependent upon attribute X of Relation R if and only if for each value of X in R has associated with it only one value of Y in R at any given time. It is represented as X->Y where X attribute is known as determinant and Y attribute is known as determined.

A Determinant is an attribute in a Relation which can uniquely determines the value of other attribute.

Let us now consider an example to explain functional dependency. We take an example of STUDENT table.

STUDENT TABLE

Rollno	S_Name	Course	S_phone	City
2405	Anshu	MCA	2225070	Amritsar
3162	Pankaj	B.Tech	2151517	Ludhiana
2789	Rakesh	MCA	2150707	Jalandhar
4162	Geeta	M.Tech	3190009	Ludhiana
3157	Pankaj	MCA	3200320	Bhatinda

In the above table, S_Name, Course, S_phone, City are functionally dependent on Primary key Rollno because corresponding to each student‟s roll number there exist one value of S_Name, Course, S_phone, City respectively which is represented as

Rollno $\longrightarrow$ **S_Name** **Rollno** $\longrightarrow$ **Course**

Rollno $\longrightarrow$ **S_phone** **Rollno** $\longrightarrow$ **City**

FEATURES OF FUNCTIONAL DEPENDENCY

The following are the features of functional dependency:

- If a attribute (column) acts as a Primary key the all the columns in the Relation (table) must be functionally dependent on the Primary key attribute.
- If X->Y in Relation R then Y->X may or may not hold.

FULLY FUNCTIONAL DEPENDENCY

Fully Functional dependency usually applies to tables with composite keys i.e. when the Primary key consists of more than one field.

An attribute (column) Y in relation R is fully functionally dependent on an attribute X of Relation R if it functionally dependent of X and not functionally dependent on any subset of X. In other words, fully functionally dependent means that when the Primary key is composite i.e. made of two or more columns of the Relation (table) must be identified by the entire key and not by some of the attributes (columns) that make up the Primary key.

Let us consider the example of ORDER_BOOK Relation holding details about all books ordered by shopkeeper.

Order_no	B_name	Quantity	B_price
4253	Learn C	15	Rs. 175
4253	Database	20	Rs. 225
4154	IT	30	Rs. 200
4256	Learn C	50	Rs. 175
4186	Database	15	Rs. 225

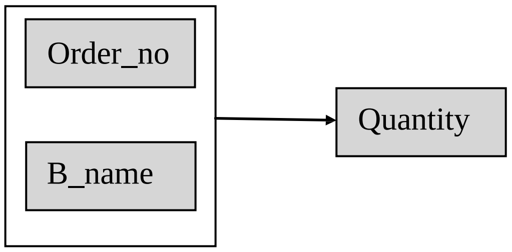
ORDER_BOOK Relation

In this table, primary key is composed of two attributes (Order_no, B_name). The attribute **Quantity** is fully functionally dependent on Primary key and not any subset of it (i.e. Order_no attribute alone or B name alone). The quantity is not functionally dependent on either **Order_no** nor on **B_name** because corresponding to each Order_no or B_name there are multiple values for the **Quantity** attribute. So it is represented as

(Order_no, B_name)->Quantity

But the B_price attribute is not fully functionally dependent on Primary key (Order_no, B_name) because B_price attribute is functionally dependent on B_name attribute (Each book is having fixed price which doesn‟t vary). Although it is not functionally dependent on Order_no attribute. But according to the definition, the attribute must not be functionally dependent on any subset of the primary key fields composed of more than one attribute.

The dependency diagram of the ORDER_BOOK table is



SECOND NORMAL FORM

A Relation is said to be in Second Normal Form (2NF) if both conditions hold simultaneously.

- The Relation is in First Normal Form (1NF)
- Every Non-key attribute (i.e. attribute(s) which don‟t form primary key) should be fully functionally dependent on the primary key.

A Relation is said to be in 2NF if it is in 1NF and no non-key attributes of the Relation should be functionally dependent on part of the concatenated primary key (i.e. primary key with multiple attributes). Instead every non-key attribute(s) should be fully functionally dependent on the primary key.

NOTE: If a Relation in the 1NF consists of only one attribute as a primary key than the Relation is said to be in 2NF

Order_no	B_name	Quantity	B_price
4253	Learn C	15	Rs. 175
4253	Database	20	Rs. 225
4154	IT	30	Rs. 200
4256	Learn C	50	Rs. 175
4186	Database	15	Rs. 225

ORDER_BOOK Relation (Fig I)

To explain 2NF, let us again consider the ORDER_BOOK Relation as shown in Fig I. Here primary key is composed of combination of attributes (**Order_no, B_name**) and the non key attributes are **Quantity** and **B_price**. Although the ORDER_BOOK Relation is in 1NF but it suffers from Insertion, Deletion, and Updation anomalies which we have already discussed.

The reason for such anomalies is because of some attributes are functionally dependent on a part of primary key rather than the whole of it. This results in Redundancy of data. The 2NF removes these partial functional dependencies hence the redundancy of data.

In the ORDER_BOOK Relation the dependencies that exist are

(Order_no, B_name)->Quantity

B_name->B_price

Here the **Quantity** attribute is fully functionally dependent on the concatenated primary key (**Order_no, B_name**). On the other hand, the **B_price** is not fully functionally dependent on the concatenated primary key because **B_price** is functionally dependent on **B_name** only i.e. it is dependent on part of concatenated Primary key.

(**Order_no, B_name)->B_price**

The steps for converting a Relation in 1NF to be a Relation in 2NF can be performed by splitting the source Relation into two or more Relations where each Resultant Relation no longer has any partial key dependencies. So to achieve this

- Create a New Relation from source Relation that contains the attributes that are fully functionally dependent on the concatenated Primary key and Primary key itself.
- Create other Relation from source Relations that contains the attributes that are not fully functionally dependent on the Primary key and Members of Primary key on which they are dependent or in the other words the Relation will include the non-key attribute(s) and the part of concatenated primary key on which it is functionally dependent. The primary key of the Resulting table will be the part of concatenated primary key on which it is functionally dependent.

ANOMALIES IN SECOND NORMAL FORM (2NF)

Even if the Relation is in 2NF it still suffers from insertion, deletion and updation anomalies. To discuss the various anomalies we will consider the STUDENT Relation that hold information about students and their teachers.

Stu_Id	Stu_Name	Teach_Id	Teach_Name	Teach_Qual
2523	Anurag	T001	Navathe	Ph. D
3712	Pankaj	T004	Date	M.Tech
4096	Gagan	T001	Navathe	Ph. D
2716	Anshu	T004	Date	M.Tech
1768	Harman	T009	Desai	M.Tech

STUDENT Relation

In this table, Stu_Id is the primary key which acts as the roll number of the student.

Since the STUDENT Relation is composed of only one attribute which acts as a primary key (Stu_Id) so it is in 2NF. But it suffers from the Insertion, Deletion and Updation anomalies which are explained as follows.

INSERTION ANOMALY

Suppose that we want to insert a tuple (record) with some information about a new teacher who has not a yet been assigned a personal student. But this insertion of tuple is not allowed because the Primary key **Stu_Id** contains a null value which is not possible as it is against the Entity Integrity rule.

Stu_Id	Stu_Name	Teach_Id	Teach_Name	Teach_Qual
2523	Anurag	T001	Navathe	Ph. D
3712	Pankaj	T004	Date	M.Tech
4096	Gagan	T001	Navathe	Ph. D
2716	Anshu	T004	Date	M.Tech
1768	Harman	T009	Desai	M.Tech
NULL	NULL	T007	Vikas	Ph. D

FIG. STUDENT Relation

DELETION ANOMALY

Suppose that a student whose Stu_Id=1768 decides to leaves the college, so we would have to delete this tuple from the STUDENT Relation. On deleting this tuple the information about the Teacher would also be deleted. This may lead to loss of vital information. This is deletion anomaly.

UPDATION ANOMALY

The 2NF also suffers from Updation anomaly.

For example: The value of the qualification of the teacher i.e. **Teach_Qual** whose **Teach_Id='T004'** is updated from **M.Tech** to the **Ph. D.** This would be quite a big problem as the updation in the tuple will have to be made where ever this information reoccurs. In case of huge databases it will be big problem and may lead to inconsistencies as human are prone to errors.

TRANSITIVE DEPENDENCY

Before discussing the third Normal Form (3NF) we should be aware of the transitive dependency. Suppose there exists a Relation R with the attributes A, B and C. A is the primary key and B and C are the non key attributes.

We assume that following dependencies hold

$A \rightarrow B$ but $B \not\rightarrow A$ (i.e. B attribute is functionally dependent on A)

$B \rightarrow C$ (i.e. C attribute is functionally dependent on B)

then C is transitively or indirectly dependent on A i.e. **A C (transitively)**

$$\begin{tikzcd}
A \arrow[r] & B & & \\
& B \arrow[r] & C \arrow[r, Rightarrow] & A \arrow[r] & C
\end{tikzcd}$$

This type of dependency is also known as Indirect dependency. Here, the dependency **C B** is neither expressively prohibited nor required by definition of Transitive dependency.

Let us now consider an example of STUDENT Relation to explain transitive dependency. In this relation the **Stu_Id** i.e. Roll number of the student is the primary key as it uniquely determines each record of the Relation. The non key attributes consist of Stu_Name, Teach_Id, Teach_Name, Teach_Qual.

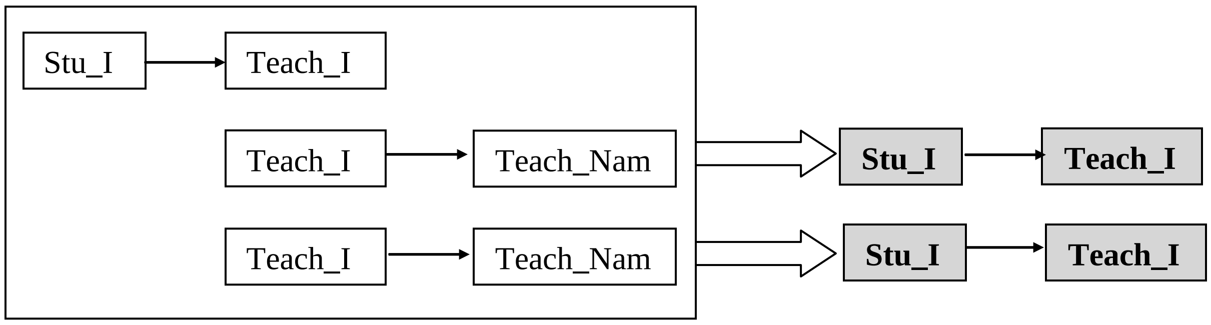
In the STUDENT Relation FIG the following functional dependencies hold.

Stu_Id $\rightarrow$ **Stu_Name** **Stu_Id** $\rightarrow$ **Teach_Id**

Teach_Id $\rightarrow$ **Teach_Name** **Teach_Id** $\rightarrow$ **Teach_Qual**

We can see from the above dependencies that **Teach_Id** is functionally dependent on the dependent on the primary key **Stu_Id** and both **Teach_Name** and **Teach_Qual** are functionally dependent on **Teach_Id.**

So it is clear that both the attributes Teach_Name and Teach_Qual are transitively functionally dependent on the primary key Stu_Id in other words they are indirectly dependent.



The concept of transitive dependence will be helpful in explaining the Third Normal Form (3NF)

THIRD NORMAL FORM (3NF)

A Relation is said to be Third Normal Form (3NF) if both condition hold simultaneously:

- The Relation is in Second Normal Form (2NF)

- Non-key attribute of the Relation should not be transitively functionally dependent on the primary key.

In other words a Relation is said to be in 3NF if it is in 2NF and every non-key attribute is fully and directly dependent on the primary key.

The main purpose of the 3NF is to remove the transitive dependency which is the main cause to anomalies in the 2NF.

So to make the Relation in 3NF we have to eliminating this transitive dependence on the primary key. The Reduction of 2NF Relation into 3NF consists of splitting the 2NF into appropriate Relations such that every non-key attribute are functionally dependent on the primary key not transitively or indirectly of the respective Relations.

For example:

In the STUDENT Relation, the non-key attributes **Teach_name** and **Teach_Qual** all transitively functionally dependent on the Primary key **Stu_Id**. Also these attributes are functionally dependent on an attribute **Teach_Id**.

So the other resulting relation in 3NF would contain **Teach_Id**, **Teach_Name** and **Teach_Qual** attributes. The primary key in this table will be **Teach_Id.**

Stu_Id	Stu_Name	Teach_Id	Teach_Name	Teach_Qual
2523	Anurag	T001	Navathe	Ph. D
3712	Pankaj	T004	Date	M.Tech
4096	Gagan	T001	Navathe	Ph. D
2716	Anshu	T004	Date	M.Tech
1768	Harman	T009	Desai	M.Tech

STUDENT Relation

Stu_Id	Stu_Name	Teach_Id
2523	Anurag	T001
3712	Pankaj	T004
4096	Gagan	T001
2716	Anshu	T004
1768	Harman	T009

STU_TEACH Relation

Teach_Id	Teach_Name	Teach_Qual
T001	Navathe	Ph. D
T004	Date	M.Tech
T009	Desai	M.Tech

TEACHER Relation

BOYCE-CODD NORMAL FORM (BCNF)

A BCNF is a stronger definition of 3NF. Unlike the 3NF where a Relational table consists of only one candidate key, the BCNF deals with Relational tables that have:-

a) Multiple Candidate Keys i.e. more than one.

b) Composite Candidate keys i.e. concatenated keys.

c) Candidate keys that are overlapped i.e. two or more of the Candidate keys share a common attribute.

In other words, for a relation to be in BCNF, a relation must only have Candidate keys as determinants. If a relation (table) contains only one candidate key, the 3NF and BCNF are equivalent. In other words, BCNF can be violated only if Relation contains more than one Candidate key.

GRADE TABLE

SName	Stu_Id	Course	Grade
Rajesh	5705	Computer	A
Kapil	5901	Computer	C
Sunil	5658	Punjabi	A
Rajesh	5705	French	A
Ankit	5816	Dance	B
Sunil	5658	Yoga	B

The Grade relation is in 3NF since there is no transitive dependencies but not in BCNF because

a) It consists of Multiple Candidate keys (SName, Course) and (Stu_Id, Course).
b) The Candidate keys in the Relation are composite keys i.e. it consists of more than one attribute like (SName, Course) and Stu_Id, Course).
c) Both Candidate keys share a common attribute „Course".

This relation has a disadvantage in the form of repetition of data which may result in Insertion, Updation and Deletion Anomalies.

INSERTION ANOMALY- You cannot insert information about a student unless he has registered with some course.

UPDATION ANOMALY- Since the **SName** and **Stu_Id** is repeated multiple times so any change in one of these would result in multiple updations. This results Inconsistency of data.

DELETION ANOMALY- If the student withdraws from all the courses he is registered the information about the student is completely lost.

The above anomalies present in Relation GRADE is due to the overlapping candidate keys (SName, Course) and (Stu_Id, Course). To remove the above problems we decompose the GRADE relation into two Relations in two possible ways

GRAD_ST(@SName+@Stu_ID) and GRAD_COU (@Course+@Stu_Id+Grade)

OR

GRAD_ST(@SName+@Stu_ID) and GRAD_COU (@Course+@SName+Grade)

GRAD_ST

SName	Stu_Id
Rajesh	5705
Kapil	5901

GRAD_COU

Stu_Id	Course	Grade
5705	Computer	A
5901	Computer	C

Sunil	5658
Ankit	5816

5658	Punjabi	A
5705	French	A
5816	Dance	B
5558	Yoga	B

The resulting relations must hold non loss less join property i.e. whenever you join the resulting relations we get the Original Relation and no extra tuples. From the definition of BCNF it is clear that a Relation in BCNF is also in 3NF but the converse is not necessarily true.

COMPARISON BETWEEN 3NF AND BCNF

Although a Relation in 3NF and BCNF sounds to similar but they are different. A table in BCNF is also in 3NF but the converse is not true.

- In BCNF, the Relation has multiple composite Candidate keys and two or more of the candidate keys share a common attribute.
- In BCNF, on splitting the source Relation into smaller Relations may not always holds the functional dependencies that existed in the source relation. This is considered to be a major drawback of BCNF which is reason why we switch to a weaker normal form 3NF. This problem is known as Dependency preservation problem.
- BCNF is simply a stronger definition of 3NF. BCNF makes no explicitly references to first and second normal form as such, nor the concept of full and transitive dependencies.
- A Relation can be normalized both with BCNF or 3NF if a relation consists of only single candidate key.

FOURTH NORMAL FORM (4NF)

The normal forms developed so far deal with functional dependencies only. It is possible for a Relation in (3NF or) BCNF to still exhibit update, insertion and deletion anomalies. This can happen when multivalued dependencies are not properly taken care of. In order to eliminate anomalies that arise out of these dependencies, the notion of Fourth Normal Form (4NF) was developed.

A Relation R is in 4NF if it is in BCNF (or 3NF) and it contains no multivalued dependencies. In other words, a Relation (table) is in 4NF if it is in BCNF (or 3NF) and all multivalued dependencies are also functional dependencies.

To explain the concept of 4NF, let us consider the **EMPLOYEE** Relation as shown below having attributes **Empname** (Name of employee), **Equipment** and **Language**.

In this **EMPLOYEE** Relation all the three attributes act as a primary key since no single attribute can uniquely identify a tuple (record).

EMPLOYEE TABLE

Empname	Equipment	Language
Anurag	PC	English
Anurag	PC	French
Anurag	Mainframe	English
Anurag	Mainframe	French

Kapil	PC	English
Kapil	PC	French
Kapil	PC	Japanese

The Relationship between **Empname** and **Equipment** is a multivalued dependency because for each pair of (Empname, Language) values in the EMPLOYEE Relation, the associated set of **Equipment** values in determined only by **Empname** and is independent of **Language.**

Equipment $_{\text{Anurag,English}}$ = **Equipment** $_{\text{Anurag, French}}$ = **{PC, Mainframe}**

Equipment $_{\text{Kapil,English}}$ = **Equipment** $_{\text{Kapil, French}}$ = **Equipment** $_{\text{Kapil, Japanese}}$ = **{PC, Mainframe}**

Thus it implies that

Empname $\twoheadrightarrow$ **Equipment**

Similarly the Relationship between **Empname** and **Language** is a multivalued dependency which is represented as

Empname $\twoheadrightarrow$ **Language**

<u>So to transform a Relation (table) with multivalued dependencies into 4NF move each MVD pairto a new Relation which is as shown below.</u>

<u>Properties of MVD</u>

- ✓ For a relation to maintain MVD, it should have atleast 3 attributes. (i) A –>> B (ii) A –>> C
- ✓ The attributes giving rise to MVD must be independent of each other (i.e B and C).
- ✓ Functional dependency is a special case of MVD. If we rtrict the set determined by MVD to a single set, then MVD becomes FD.

<u>EMPLOYEE</u>

Empname	Equipment	Language
Anurag	PC	English
Anurag	PC	French
Anurag	Mainframe	English
Anurag	Mainframe	French
Kapil	PC	English
Kapil	PC	French
Kapil	PC	Japanese

EMP_EQUIP

Empname	Equipment
Anurag	PC
Anurag	Mainframe
Kapil	PC

EMP_LANG

Empname	Language
Anurag	English
Anurag	French
Kapil	English
Kapil	French
Kapil	Japanese

DATABASE SECURITY

Database security has to do with ensuring that only the right people get the right access to the right data. There are two basic problems here. The first deals with "right people". The right people are those who have the privilege to interact with the database. The problem of determining that the right people interact with the database is called the problem of authentication. The second basic problem in database with the part about "right access to right data ". Even though a certain user may be allowed to interact with the database, it is often required that this user can do so in controlled ways.

AUTHENTICATION

It is the problem of checking whether the user operating upon the database is the correct user for the database. Without authentication the user can access the data, which is not meant for them. It is relatively easy for the users, either inadvertently or deliberately to access data not meant for them unless special efforts are made to prevent this. These special efforts can make use of passwords which can be associated with sub schema. These are other methods of checking the authenticity of users. These can be based on special devices into which a magnetic film badge can be inserted. This film would contain a pattern which would match against a corresponding pattern stored in a machine. A more elaborate method of performing authenticationcan be based on fingerprints. Whenever an access to the database is desired, the pattern on the fingerprints of the user can be read off on the fingerprint reader and a match can be made with thestored fingerprints. Access is permitted if the fingerprints of the user match the fingerprints in the system.

Another type of check nowadays is the usage of retina scanner. A scan of the retina of the user is taken an stored in computer. Whenever an access to the database is derived, a fresh retina scan is taken. If it matches with the already stored retina pattern then the user is given access otherwise not.

- **AUTHORISATION AND ACCESS CONTROL**

Authorization is used to check the extent to which an authenticated user can interact with the database. It is basically a controlling body which limits the access of a user tothe database.

A MODEL FOR ACCESS CONTROL(LAMPSON MODEL)

Lampson developed the basic model of access control and many people have since extended it. In essence, this model defines a matrix, called the access matrix that relates three things together.

They are:

- Subjects
- Objects
- Access rights

Loosely speaking subjects correspond to the accounts user of the example above, objects to data items like salary and access rights to read or write. An access matrix would specify that the accounts user is allowed to read the data item, salary but not write into it. Now, the way to construct the access matrix is to, first, identify all the subjects, objects and access rights. Thereafter, each row of the access matrix is labeled with a different subject and each column with a different object. The slots in the matrix are filled in with the access right that the subject (corresponding to the slot) has on the object. An entry of the access matrix is shown on the next page.

Subject object	Ename	Salary	Leave
Clerk	R	-	-
P. Manager	R	-	-
Sr. Manager	**R/W**	**-**	**R/W**

- <u>Subjects</u>

  The access matrix specifies the access right that a subject may have over an object. The question is as to what can be a subject. In one case, we have seen that the different users of database are the subjects of access matrix like MD, DBA, PM etc. these users are identified in the system analysis process which is used to build a database.

- <u>Objects</u>

  An object can be any unit of the database like department name, dept. budge, employee name and it is a pattern for the policy governing database usage to identify all objects. Since the units in terms of which a database is defined vary from one model of the data to another. Objects can be areas, relations or record types set types and attributes or fields. It is also possible for program to be objects.

- <u>Access Rights</u>

  It must be noted that these has to be some correspondence between the access rights, and the operations that can be performed on database. There are two schemes for access rights structuring

  i. The first of these treats access rights as flat.

  ii. The second one treats them as hierarchically.

i. Under flat schema access rights are independent, stand alone things which used to be explicitly specified.

ii. In hierarchical schema the possession of a certain access right may imply the possession of certain other rights subordinate to it.

<u>**THE AUTHORISED**</u> In the foregoing we have assumed that there is something that there is somebody who decides which subjects have what rights on what objects. This body is known as authorizer. There seem to be two ways in which an authorizer can be nominated. These are as follows:--

a) Any subject who creates an object is the authorizer for that object. This means that body which creates an object has all rights over that object.

b) A body like DBA or the enterprise manager is explicitly entrusted with the function of the authorizer.

DATA ENCRYPTION

Data encryption and access control are the internal computer techniques used to protect the data from unauthorized discloser, alternation and destruction. The access control mechanism will not avoid all unauthorized accesses, in fact over half of all reported abuses invoking the computer resulted from insiders abusing their access authority.

The access control is ineffective if

- ✓ Password are written & found.
- ✓ Off-line backup files are stolen.
- ✓ Someone taps on a communication line.
- ✓ Confidential data is left in main memory after job has completed

So, another technique of data encryption is used for protecting the sensitive data against all unauthorized users.

Properties of good encryption techniques

Relatively simple for authorized users to encrypt and decrypt data.

- Encryption technique not only depends on the secrecy of a parameter of algorithm called encryption key.
- Extremely difficult for an intruder to determine the encryption key.

Conventional Encryption

This kind of encryption schema has following parts:--

- **Plaintext**:-

This is the original message or data that is fed into the algorithm as input.

- **Encryption algorithm**:-

The algorithm performs the various substitutions and transformations on the plain text.

- **Secret key**:-

It is also input to the encryption algorithm. The exact substitution and transformations performed by algorithm depends on the key.

- **Cipher text**:-

This is scrambled message produced as the output. It depends upon plain text & the secret key. For a given message, two different keys will produce two different cipher texts.

Original phrase:-- database management

Scrambled message:--ADATABESM NAAGMENET

Decryption Algorithm

It is just like running the encryption algorithm in reverse order. It takes the cipher text & the secret key and produces the original plain text.

Transmitted

Cipher text

Plain text---------→ Encryption---------------→ Decryption -------→ plain text

Algorithm Algorithm

Models for Data Encryption

- **Techniques used for encryption**

The traditional methods used for data encryption are the following:--

(i) Transposition ciphers (ii) Substitution ciphers

(i) Transposition ciphers:-

They retain the identity of original message or characters of plain text but change their position e.g. If the transposition rule is to transpose each consecutive pair of characters.

Original Phase:-data encryption it appears or transposition as

Cipher text:- adataeb cnyrtpoin („b‟ is blank space)

However, this rule is not very secure. So, other rules can also be devised and one of them is to take permutation to form blocks of four characters & permutate 1234 to 3214.

(ii) Substitution ciphers:-

They retain the relative position of the characters in the original plain text but hide their identity in the cipher text. In the substitution cipher each letter or group of letters is replaced by another letter or group of letters to disguise it.

Example:-- Plain Text: ENCRYPTION becomes

Cipher text: HQFUBSWLRQ

Such a cipher is very easy to break. So some other improved ciphers used are mono-alphabetic substitution ciphers, poly-alphabetic substitution ciphers etc.

Data encryption standard (DES)

In 1977 the US National Bureau of Standards adopted a standard for data encryption called DES. It was widely adopted by Industry for the use in security products.

- **Public key encryption**:- This is an another encryption technique proposed by Diffie and Hallman in 1976. Public key algorithms are based on mathematical functions rather than on simple operations, on bit patterns. A public key encryption includes:
  1) Plain text
  2) Encryption algorithm
  3) Public key and private key: Each user has two keys.

  - **Public key**

  Publicly published key used to encrypt data but cannot be used to decrypt data.

  - **Private key**

  Key known only to individual user and used to decrypt data need not be transmitted to the site doing encryption.

4) Cipher text
5) Decryption algorithm

Following diagram showing public key encryption:

```
                              Transmitted
                              Cipher text
Plain text-----→ Encryption--------------→ Decryption -------------- → Plain Text
                  Algorithm                 Algorithm           Output
                  Eg. RSA                   Eg. Reverse of encryption
                                             Algorithm
```

Public keyencryption algorithm works 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 two keys in a public register known as public key and the other key is known as private key. Each user maintains collection of public keys obtain from others.
3. If Anshu wants to sends private message to Anurag then Anshu encrypts the message using Anurag‟s public key.
4. When Anurag receives the message be decrypts it using private key. Since only Anurag knows his private key. No other recipients can decrypt the message.

The big advantage of public key encryption is that it doesn‟t require secure transmission of keys between sender and receiver.

Disadvantages of Encryption:--

1) Encryption of data gives use to serious technical problems at the level of physical storage organization. Eg. Indexing over data, which is stored in encrypted from can be very difficult.

2) Keeping keys secret is a problem. As long as a user protects his private key incoming communication is secure otherwise intender can decrypt it.

3) Breaking cipher text and discovering a key is often easier if the intruder can obtain some plain text and its corresponding cipher text.

CONCURRENCY CONTROL

Problem

The problem of concurrency control deals with ensuring that two or more users do not get into each other‟s way. It must be noted that problems of concurrency occurs only when at least one of the concurrent users wishes to update the database. When all the concurrent users are only readers of the database, no difficulties arise. There are three ways in which users can interface with each other and which give rise to the following problems:

- Lost updates.
- Dirty reads.
- Unrepeatable reads or inconsistent analysis.

(a) Lost updates

Consider the two users u1 & u2 of a database. Now both wish to add fifty rupees to a certain bank account. This portion of this activity relevant to us is:

Read balance.
Balance=balance+50.
Update balance.

It is possible that they be executed in a sequence like

U1: read balance.
U1: balance= balance+50.
U1: update balance.
U2: read balance.
U2: balance= balance+50.
U2: update balance.

If this order is needed followed, the balance is properly updated by a hundred rupees. On the other hand, some of the operations performed by u1 may get interleaved with those performed by u2 as shown below:-

U1: read balance
U2: read balance
U1: balance= balance+50
U2: balance= balance+50
U1: update balance
U2: update balance

After the transactions, we have the following details for U! and U2

U1=1050
U2=1050

The net result is that the balance is updated by fifty rupees & not by a hundred! Clearly, one of the two updates has been lost and we say that a lost update has occurred.

There is another way in which the lost updates can surface. Consider, again two users u1 & u2.

U1: update x

U2: update x

U1: backup

Here u1& u2 updates the same item. Therefore, u1 decides to undo its action & backs up to its original state. Clearly the update performed by u2 shall get lost.

(b) <u>Problem of dirty reads</u>

Assume two users U1 & U2. Let them perform the following operations:-

U1: update x

U2: read x

U1: abort action

Here u2 reads a value that has been updates by u1. however u1 aborts its action. Clearly, whatever has been read by u2 is meaningless. This is known as the problem of dirty reads.

(c) <u>Unrepeatable reads</u>

Lastly, consider the problem of unrepeatable reads. Let the sequences of execution be:

U1: read x

U2: update x

U1: read x

Here u1 reads the value of x, both before & after u2 has updated it. In general, the two values are likely to be different. Any analysis that u1 makes with these two values is likely to be inconsistent.

<u>Schedule</u>

The order is which the concurrently running transactions are executed is called a schedule. The schedules are classified as serial and non- serial schedules.

<u>Types of schedules:--</u>

The basic of the problem of concurrency lies in that each user of a database writes programs or queries under the assumption that there is no other user. Therefore, whenever we have a schedule which causes all the actions of the transactions are taken up for execution, we get satisfactory results. However, when two transactions enter execution simultaneously, the order of the execution of the action is determined by the computer system.

Assume two transactions T1 & T2 each of which transfers money from one account to another. Let the account be X, Y & Z. let the balances in X, Y & Z be Bx, By & Bz. Respectively. The two transactions are as follows:

T1	T2
Read Bx	Read By
Bx=Bx-50	By= By-60

Update Bx	Update By
Read By	Read Bz
By= By+50	Bz= Bz+60
Update By	Update Bz

Consider the following three schedules:--

Schedule 1	**Schedule 2**	**Schedule 3**
T1: Read Bx	T1: Read Bx	T1: Read Bx
T1: Bx=Bx-50	T1: Bx=Bx-50	T1: Bx=Bx-50
T1: Update Bx	T1: Update Bx	T2: Read By
T1: Read By	T2: Read By	T2: By= By-60
T1: By= By+50	T2: By= By-60	T1: Update Bx
T1: Update By	T2: Update By	T1: Read By
T2: Read By	T1: Read By	T1: By= By+50
T2: By=By-60	T1: By= By+50	T1: Update By
T2: Update By	T1: Update By	T2: Read Bz
T2: Read Bz	T2: Read Bz	T2: Bz=Bz+60
T2: Bz=Bz+60	T2: Bz=Bz+60	T2:Update Bz
T2:Update Bz	T2: Update Bz	T2:Update By
Serial schedule	**Serializable Schedule**	**Non- Serializable Schedule**

The first schedule is a serial schedule. That is, all the actions of T1 occur before those of T2. the second schedule is not serial but it can be verified that it presents no difficulties. Such a schedule which is not serial but behaves like one is called serializable schedule. The last schedule is a problematic schedule. In this schedule update of By to By+50 gets lost.

Clearly, any schedule that is either a serial one or a serializable one is a good schedule. There are two ways by which non- serializable schedule can be handled. The first tries to prevent a non- serializable schedule from even occurring. It relies on the notion of locks. The second way is the non- serializable schedule detector. Here, the attempt is to detect a situation which shall lead to a non- serializable schedule & then to make the offering transactions withdraw & start all over again.

<u>Locks</u>

The notion of locks helps us to convert a schedule into a serializable schedule. A lock can be placed by transaction on a resource(file, data etc.) that is it desires to use. When this is done, the resource is available for use exclusively to that transaction. In this sense, other transactions are locked out of the resource. When a transaction that has locked a resource does not desire touse it anymore, it should unlock the resource so that other transactions can use it.

T1	**T2**
Lock Bx	Lock By
Lock By	Lock Bz

Read Bx	Read By
Bx=Bx-50	By=By-60
Update Bx	Update By
Unlock Bx	Unlock By
Read By	Read Bz
By=By+50	Bz=Bz+60
Update By	Update Bz
Unlock By	Unlock By

It can be seen that a schedule comprising T1 & T2 shall be serializable.

The notion of locks as outlined above is quite restrictive. It turns out that a greater amount of concurrency is achieved if locks are classified as shared or exclusive. If a transaction is one which never updates the database, it shall have no difficulty in co-existing with another which also never updates the database. In such a situation, the transaction, when locking a resource, places a shared locked on it. On the other hand, **if a transaction is an updating transaction, it has to ensure that no other transaction can access the resource it is updating**. In this case, the transaction places an **exclusive lock** on the resource. The properties of shared & exclusive locks are summarized below:

<u>Shared Lock</u>

This should be used by a transaction which is a read only transaction for a resource. A shared lock does not allow an exclusive lock to be placed on the resource but permits any number of shared locks to be placed on it.

<u>Exclusive Lock</u>

This should be used by a transaction which shall perform an update operation on a resource. No other transaction can place either a shared lock or an exclusive lock on a resource that has been acquired in an exclusive mode.

Consider now the situation where there exists two banking accounts A & b. let their balances be Rs.1000 & Rs. 900 respectively. Consider the two transactions produce the sum ofthe balances in account A & B and other transaction transfers a Rs 200 from account A to accountB.

T1	**T2**
Lock X(A)	Lock X(Sum)
Read(A)	Sum:=0
A:=A-200	Lock S(A)
Write(A)	Read(A)
Unlock(A)	Sum:=Sum+A
Lock X(B)	Unlock(A)
Read(B)	Lock S(B)
B:=B+200	Read(B)
Write(B)	Sum:=Sum+B

Unlock(B)	Write(Sum)

Schedule

T2:Lock(Sum)

T2:Sum:=0

T2:Lock S(A)

T2:Read(A)

T2:Sum:=Sum+A

T2:Unlock(A)

T1:Lock X(A)

T1:Read(A)

T1:A:=A-200

T1:Write(A)

T1:Unlock(A)

T1:Lock X(B)

T1:Read(B)

T1:B:=B+200

T1:Write(B)

T1:Unlock(B)

T2:Lock S(B)

T2:Read(B)

T2:Sum:=Sum+B

T2:Write(Sum)

T2:Unlock(B)

T2:Unlock(Sum)

This schedule is a non-serializable schedule and a non serializable schedule has occurred even though both the transactions locked and unlocked the data items used by them.

Two- Phase Locking(2PL)

Locking may help in achieving serializibility but it does not always guarantee it. One way of guarantee serializability is to use, an additional protocol concerning with the positioning of locking and unlocking in every transaction known as two phase locking(2PL).

A transaction is said to follow the two phase locking protocol if all locking operations precede the first unlock operation in the transaction. Each transaction in 2PL issues lock and unlock request in two phases:

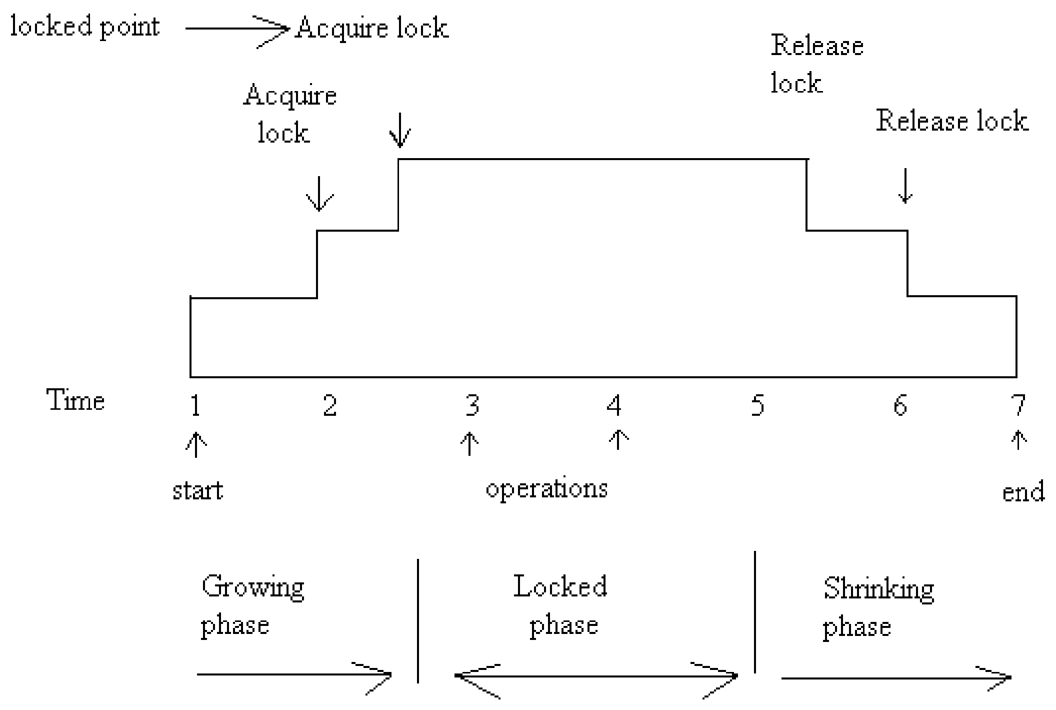
1. Growing phase
2. Shrinking phase

Growing phase

The growing phase is also known as expanding or first phase. In this phase the transaction acquires all locks needed but cannot release any locks.

Shrinking phase

The shrinking phase is also known as second phase or contracting phase. In this phase, the transaction releases the existing locks but cannot acquire new locks.



SERIAZIBILITY

Construction of a Dependency Graph to detect Serializability

An algorithm can be devised to answer whether a schedule is serialisable or not. The idea is to construct a dependency graph which has the transactions of the schedule represented as its nodes. This graph has directed edges. **An edge from the transaction T_i to T_j means that T_j depends upon T_i.** That is, there exists an a ction of T_j which takes some of its inputs from an earlier action T_i. It has been proven that a serialisable schedule is such that its dependency graph contains no cycles. **If the dependency graph contains no cycles in it, then we have a serial schedule corresponding to it.**

Steps to construct the dependency graph

1. For every transaction in the schedule, **create a node** in the graph.

2. For a transaction T_i , which has shared locked an item A, find a transaction T_i which shall next exclusively lock the item A. **Draw an edge from T_i to T_j provided T_j is found.**

3. For a transaction T_i, which has exclusively locked an item A, find a transaction T_j which shall next exclusively lock A. If such a transaction is found, draw an edge from T_i to T_j.

4. Now, look at transactions falling between T_i and T_j to determine if there is a transaction T_k which shared locks A after T_i has unlocked it and before Tj has locked it. **If no T_j was found, then T_k is transaction that shared locks A after T_i has exclusively locked it.** Draw an edge from T_i to T_k for every T_k found.

If the graph constructed through steps 1 to 4 has no cycles in it, then the schedule which it represents is serialisable.

Consider the following schedule:

T_1: Lock shared A

T_2: Lock exclusive B

T_1: Unlock A,

T_3: Lock exclusive A

T_2: Unlock B

T_1: Lock shared B

T_3: Lock shared B

T_3: Unlock A

T_2: Lock exclusive A

T_2: Unlock A

T_1: Unlock B

T_3: Unlock B

To see how-the graph of Fig.1-is constructed, startby looking for the first unlock statement in the schedule. This happens for transaction T_1 which unlocks A. T_1 had earlier shared locked A. The next time it is locked by T_3 in exclusive mode. **By Rule 2, and an edge is drawn from T_1 to T_3. By Rule 3, an edge should be drawn from T_3 to T_2.**

Now, consider the execution sequence on B. **It is first locked by T_2 in the exclusive mode, after which there is no transaction which locks B in exclusive mode. However, T_1 and T_3 lock it in the shared the mode.** By Rule 4, an edge is drawn from T_2 and T_1, and another from T_2 to T_3. With this, all resources operated upon in the schedule are considered and the graph of the Fig.1 results.

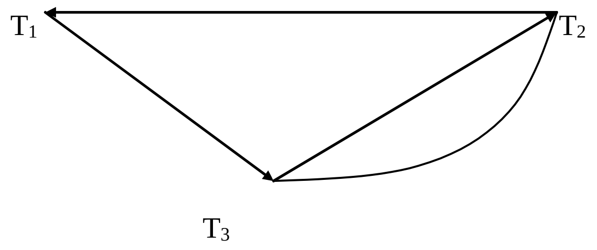


Fig. 1

<u>The dependency graph for the above schedule has a cycle in it, therefore it is non-serialisable.</u>

<u>Time Stamps Based Algorithm to detect Serializability</u>

The basis for this technique is that of imposing a linear order on the transactions, The argument is that if these transactions were to execute in this order, a schedule composed of these would be serial. Assume that there are two transactions T1 and T2 such that T1 precedes T2in the order. Then, the conditions which must hold for a schedule are:

- ✓ T2 can perform any operation (read or write) after an operation has been performed by T1.
- ✓ T1 cannot read an item that has been written upon by T2.
- ✓ T1 cannot write an item that has been read by T2.
- ✓ T1can read an item that has been read by T2

The way to implement a non-serialisable schedule detection algorithm is to stamp each-transaction with the current time of the machine clock. That is, **each transaction is associated with the time at which it was taken up for execution. Since the time of the clock is serially increasing we are sure that the transactions are serially ordered.**

Now, we associate with a resource A two marks, the read mark, Ar, and the write mark, Aw. These two marks contain the time stamp of the transaction with the highest time-stampso far to have read the resource or to have written into the resource respectively.

For example, consider two transactions with time-stamps 100 and 150 respectively. Let the first of these read A. Then, the highest transaction to have read A is the one with time stamp 100. Therefore, Ar is 100. Now, if A is read by the second transaction again, Ar becomes 150. Thereafter, if A is read by the first transaction again, Ar remains 150, since the transaction with the highest time-stamp to have read A so far is the second one.

It is now a simple matter to restate the conditions for a serial schedule outlined earlier. These are:

1. A transaction with time-stamp i cannot perform the read operation on a resource X if Aw> i
2. A transaction with time-stamp i cannot perform the write operation on a resource X if Ar> i.

Any transaction violating these conditions should be aborted and restarted.

T1 (timestamp 100)	T2 (timestamp 150)	Ar	Aw	Br	Bw
Read A		100	0	0	0
A = A-100		100	0	0	0
Read B		100	0	100	0
	Read A	150	0	100	0
	Read B	150	0	150	0
	A = A-100				
	B=B+100				
	Write A	150	150	150	0
	Write B	150	150	150	150
B - B+100		150	150	150	150
Write A			clash	150	150
Write B					

LOCK GRANULARITY

A database is basically represented as a collection of named data items. The size of the data item chosen as the unit of protection by a concurrency control program is called granularity. Granularity can be a field of some record in the database, or it may be a larger unit such as record or even a whole disk block. Granule is a unit of data individually controlled by the concurrency control subsystem. Granularity is a lockable unit in a lock-based concurrency control scheme. Lock granularity indicates the level of lock use. Most commercial database systems provide a variety of locking granularities. Lock can take place at the following levels:

- Database level.
- Table level.
- Page level.
- Row (tuple) level.
- Attr[i]butes (fields) level.

Database Level Locking

At database level locking, the entire database is locked. Thus, it prevents the use of any tables in the database by transaction T2, while transaction T1, is being executed.

Database level of locking is suitable for batch processes. Being very slow, it is unsuitable for on-line multi-user DBMSs.

Table Level Locking

At table level locking, the entire table is locked. Thus, it prevents the access to any row (tuple) by transaction T2 while transaction T1 is using the table. If a transaction requires access to several tables, each table may be locked. However, two transactions can access the same database as long as they access different tables.

Table level locking is less restrictive than database level. But, it causes traffic jams when many transactions are waiting to access the same table. Such a condition is especially problematic when transactions require access to different parts of the same table but would not interfere with each other. Table level locks are not suitable f or multi-user DBMSs.

Page Level Locking

At page level locking, the entire disk-page (or disk-block) is locked. A page has a fixed size such as 4 K, 8 K, 16 K, 32 K and so on. A table can span several pages, and a page can contain several rows (topics) of one or more tables.

Page level of locking is most suitable for multi-user DBMSs.

Row Level Locking

At row level locking, particular row (or tuple) is locked. A lock exists for each row in each table of the database. The DBMS allow s concurrent transactions to access different rows of the same table, even if the rows are located on the same page.

The row level lock is much less restrictive than database level, table level, or page level locks. The row level locking improves the availability of data. However, the management of row level locking requires high overhead cost.

Attribute (or Field) Level Locking

At attribute level locking, particular attribute (or field) is locked. Attribute level locking allows Concurrent transactions to access the same row, as long as they require the use of different attributes within the row.

The attribute level lock yields the most flexible multi-user data access. However, it requires a high level of computer overhead.

LOCK TYPES

The DBMS mainly uses the following types of locking techniques:

- Binary locking.
- Exclusive locking.
- Shared locking
- Two-phase locking (2PL)

Binary Locking

In binary locking, there are two states of locking namely (a) locked (or '1') or (b) unlocked ('0'). If an object of a database table, page, tuple (row) or attribute (field) is locked by a transaction, no other transaction can use that object . A distinct lock is associated with each database item. If the value of lock on data item X is 1, item X cannot be accessed by a database operation that requires the item. If an object (or data item) X is unlocked, any transaction can lock the object for its use. As a rule, a transaction must unlock the object after its termination. Any database operation requires that the affected object be locked. Therefore, every transaction requires a lock and unlock operation for each data item that is accessed. The DBMSs manages and schedules these operations.

Two operations, lock_item(data item) and unlock_item(data item) are used with binary locking. A transaction requests access to a data item X by first issuing a lock_item(X) operation. If LOCK(X) = 1, the transaction is forced to wait. If LOCK(X) = 0, it is set to 1 (that is, transaction locks the data item X) and the transaction is allowed to access item X. When the transaction is through using the data item, it issues unlock_item(X) operation, which sets LOCK(X) to 0 (unlocks the data item) so that X may be accessed by other transactions. Hence, abinary lock enforces mutual exclusion on the data item.

Shared/Exclusive (or Read/Write) Locking

A shared/exclusive (or Read/Write) lock uses multiple-mode lock. In this type of locking, there are three locking, operations namely (a) Read_ lock(A), (b) Write_ lock(B). and Unlock(A). A read-locked item is also, called share-locked, because other transactions arc allowed to read the item.

A write-locked item is called exclusive lock because a single transaction exclusively holds the lock on the item. A shared lock is denoted, by S and the exclusive lock is denoted by X.

A share/executive lock exists when access is specificially reserved for the transaction that locked the object. The exclusive lock must be used when there is a chance of conflict.

An exclusive lock is used when a transaction wants to write (update) a data item and no locks are currently held on that data item by ally other transaction. If transaction T2 updates data item A, then an exclusive lock is required by transaction T2 over data item A. The exclusive lock is granted and only if no other locks are held on the data item.

A shared lock exists when concurrent transactions are granted READ access oil the basis of a common lock. A shared lock produces no conflict as long, as the concurrent transactions are Read-only. A shared lock is used when a transaction wants to Read data from the database and no exclusive lock is held on that data item. Shared locks allow several READ transactions to concurrently Read the same data item. For example if transaction T1 has shared lock on data item A, and transaction T2, wants to Read data item A, transaction T2 may also obtain a shared lock on data item A.

Two-phase Locking (2PL)

Two-phase locking (also called 2PL,) is a method or a protocol of controlling concurrent processing in which all locking, operations precede the first unlocking operation. Thus, a transaction is said to follow the two-phase locking protocol if all locking operations (such as read_lock, write_lock) precede the first unlock operation in the transaction. Two-phase locking is the standard protocol used to maintain level-3 consistency 2PL defines how transactions acquire and relinquish locks. The essential discipline is that after a transaction has released a lock it may not obtain any further locks. In practice this means that transactions hold all their locks they are ready to commit. 2PL has the following two phases:

- A growing phase, in which a transaction acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
- A shrinking phase, in which a transaction releases all locks and cannot obtain any new lock.

The above two-phase locking is governed by the following rules:

- Two transactions cannot have conflicting locks.
- No unlock operation can precede a lock operation in the same transaction.
- No data are affected until all locks are obtained, that is, until the transaction is in its locked point.

RECOVERY

When a database does not satisfy consistency requirements, it has to be brought into a consistency state. The techniques which do this are called recovery techniques. The kinds of inconsistencies that arise may be those that occur when semantic integrity constraints specified in the schema are not satisfied. In addition , these inconsistencies may arise out of the violation of certain implicit constraints that are expected to hold for a database.

It shall be assumed that a program which updates the database and which terminates normally does not leave the database in an inconsistent state. In other words, a program is assumed to be semantically correct. It is, therefore, only in the case of failure of programs to terminate normally that a database is in danger of being in a corrupted state. It is only in such cases that recovery techniques apply.

DIFFERENT KINDS OF FAILURES

When a program goes into execution a number of difficulties can arise, causing it to terminate abnormally. They are:-

a) Failure of statement. (b) Failure of program. (c) Failure of the system.

a) Failure of statement:- A statement of a program may cause abnormal termination if it does not execute completely. e.g. During the execution of statement , an integrity constraint may get violated and the Program may have to be aborted. If this is done, however any updates already performed may have got reflected in database leaving it in a corrupt state.

b) Failure of program:- Failure of program can occur in a number of different situations. Consider example- a program which goes into an infinite loop. The only way to break the loop is to abort the program. However this may leave the data base in an inconsistent state, since the part of the program already executed may have caused a partial update to the database.

c) Failure of the system: Failure can range from environmental & hardware failures of the failures in the system software like the operating system or the data base management system being used. As an example of an environmental failure, consider a voltage flicker in the power supply to the computer which makes it go off. As a hardware failure consider the possibility of controller malfunction & as an example of a system software failure consider the system going into a hung state.

CONTROLLING FAILURES

Even though recovery techniques have been devised to handle failures, they can be quite expensive, both in time & in memory space. Every attempt at recovery is non-productive in that if the failure had never occurred, the recovery need not have been done & the computer could have been put to some constructive use.

Consider system failures. For controlling failures from environmental factors like poor quality of power, there is a host of electrical gadgetry which can be used. Eg. One could use voltage stabilizers, isolation transformers & even an uninterrupted power supply. For hardware failures of all kinds, it is best to carry out all preventive maintenance schedules as rigorously as possible. Failure of system software can be controlled by ensuring that all known patches have actually been placed in position & by reporting all buds and other difficulties to the software vendor, so that approximate solutions can be supplied. This ensures that the problem does not repeat in the future.

Failure of a statement can arise as a result of the statement violating the semantic integrity constraint or because some of its preconditions are not satisfied. This is, of course, under the assumption that there is no system malfunction during the execution of the statement.

- Checking of integrity constraints, whenever possible is done before the statement goes into the execution and
- The preconditions to be satisfied by a statement are all checked out before the statement is executed.

Lastly consider controlling failures of programs, if program fails because of the deadlock handling strategy or the serialization handling strategy.

RECOVERY TECHNIQUES

When doing recovery, there are three broad steps to be followed. They are:-

- Collect recovery data
- Determine that recovery is needed
- Perform recovery

There are two main strategies for performing recovery. They are:-

- Backward recovery
- Forward recovery

The aim of both these is to take the current database to a consistent uncorrupted state. The former strategy achieves this by taking the database back to a good state that existed before the failure actually occurred. Forward recovery strategies, on the other hand try to help the user in taking the database forward from the current state to the good state that the user would have reached in future.

Now, a good recovery system should have the following properties:-

- The amount of data lost after recovery had been performed should be minimized.
- Recovery should not take too much time.
- Recovery should be done program wise.
- Recovery data collected should be safe and should not get lost or corrupted.
- As far as possible recovery should be automatic.

COLLECTING RECOVERY INFORMATION

There is large number of ways in which recovery data can be collected. We shall consider here.

- Database dumps (also known as database backup or as image copies)
- Save points
- Journals (also known as logs)
- Differential files

Database dumps

A database dump is essentially a copy of database taken at some point in time. Since this requires that the entire database be copied, the operation can be quite time consuming. During the time when the dump is being taken, the database should be in quiescent state. Otherwise, the copy may be inconsistent one. This means that no user who desired to update database should be allowed to access it, while the process of taking dump is going on. It is necessary that the frequently taking the dump should be carefully controlled so as not to unnecessarily spend too much time taking them and also to ensure greater availability of the database to the user community.

Save points

Savepoints are the points in the program at which the status of the program is saved. They are user defined and it is up to the user to decide where to place save points, if any, in the program. Again a proper sequence of frequency with which save points are taken should be made.

Journals

Journals are the popular way of collecting recovery data. Frequently, the life of data on journal is from one database dump time to another. A journal entry is made every time the database is updated. It can consist of:-

a) The old value of the data i.e. the value that was updated.

b) The new value of the data.

c) Both, the old as well as the new value.

In addition, the journal shall have information about the transaction that caused the update & other such housekeeping data. This data varies from one implementation to other.

Example:- Let there be an employee named Rama earning a salary of Rs.4000. Let an update statement to modify the salary to Rs.4500 be issued. As shown in fig.1 & fig.2, initially, the database contains the data about Rama & the journal is empty. After the update is performed, the database contains the modified value of salary and the journal contains the old value as well as the new value of the data. This schema assumes schema (c) above.

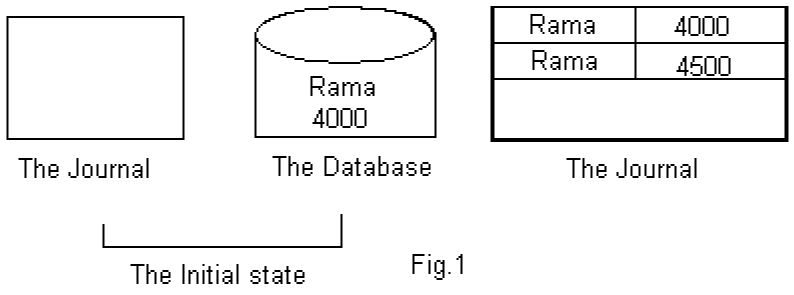


Fig.1

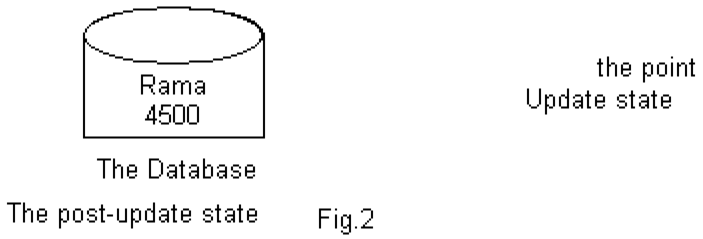


Fig.2

Differential files

Differential files are files in which changes are made to the database are recorded. Differential files can be thought of as journals which have only the new values of the data base entered but with one important difference. Whereas journals are built under the assumption that the database has had the updates entered in it, a differential file is created under the assumption that no changes to the data base have been made. That is, all changes made to the database are, in fact, trapped in the differential file. This raises the questions:

a) How does the actually get updated?

b) How is information accessed from the differential file/database combination?

In order that the database is actually updated, the contents of the differential file must get periodically merged with database. This periodicity is again a choice which the designers of the recovery schema have to make. It is possible to define this periodicity as the point at which database reorganization takes place.

Let us address ourselves to the question with all the updates trapped in the differential file, a database access not by pass this file. If indeed the database were to be bypassed the data accessed would be the one that existed the time when the differential file was last merged into the database. This obviously, would not do. So ,the database management system has to take cognizance of the fact the differential is being used and hence, route all database accesses via this file, i.e. to satisfy any access, the differential file searched first and if the access can be satisfied ,no search is performed in the database .On the other hand if the access cannot be satisfied form the differential file then and only then is a database search performed. It can therefore, be seen that the differential rarely an extension of the data base in so far as obtaining latest information is concerned .

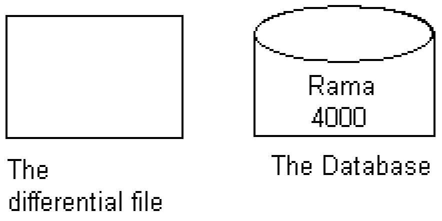


Fig.1

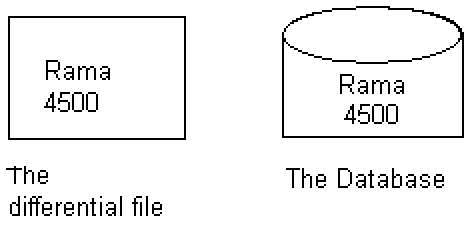


Fig.2

SYSTEM LOG

Transaction represents the basic unit of recovery in the database system. The system must maintain a log to recover from the failures that effect transactions so as to keep track of all transaction operations that effect the values of database items. The system log is also known as DBMS journal. It is usually written to stable storage and contains the redundant data required to recover from volatile storage failures and also from errors discovered by transactions.

Since it is usually maintained in the disk so it is not effected by any type of failure except for disk or catastrophic failures (natural disasters). To guard against catastrophic failures a log file is periodically backed up to archival storage devices such as magnetic tapes, optical disks etc. .

A log consists of sequence of log records which maintain a record of all updates activities in a database.

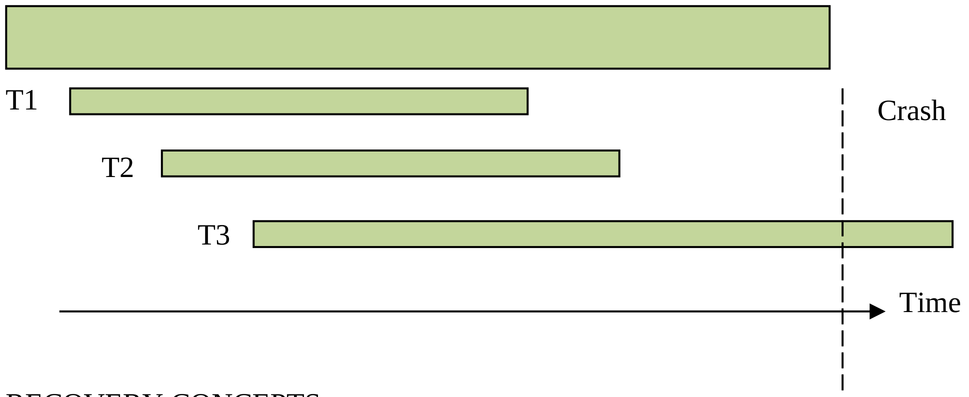
The various types of log records are as follows:

1. A start of transaction marker denoted by [Start_Transaction,T]
2. A update log record describes a write operation indicated by [Write,T, A, old value, new_ value]. Here the transaction T has performed a Write operation on data item A such as changing value of data item A from old value to new value.
3. When the transaction T is completed successfully and changes are recorded permanently to the database then it considered committed and it is represented by [commit, T]
4. When the transaction T is not completed successfully it is aborted and is represented by [abort, T]

When all operations of a transaction have been executed successfully and effect of all transaction operations on database have been recorded in the Log, the transaction is said to reach its commit point which means it is partially committed. Beyond the commit point the transaction is said to be commit and its effect is assumed to be permanently recorded in the database. The transaction then writes a commit record into the log.

If a system crash occurs then we search the log for the transactions which have written start-transaction record into the log but have not written commit record yet, then these transactions should be rolled back to undo its effect on database during recovery process.

On the other hand if the transactions have written the commit record then their effect on the database must be redone from the log records.



RECOVERY CONCEPTS

LOG BASED RECOVERY TYPES

In Log-based recovery, a log is kept on stable storage and consists of a sequence of log records. The log will record the sequence of database operations, and can be used-to replay the database actions after a failure. The recovery manager uses the logic to restore data items to their consistent state.

Recovery from transaction failures results in restoring the database to most recent consistent state prior to the failure. To do this system, uses log files to keep track of all the changes that were applied to data items by various transactions.

There are various types of failures that can occur in a database system including system crash, Logical errors, Natural disasters etc. To recover from Non natural disasters such as system we have to keep backup of data on archival storage media such as magnetic tapes etc. To recover from Non-Natural disasters such as system crash there are two main log based recovery techniques used for the recovery.

1) Immediate Update 2) Deferred Update

1. DEFERRED UPDATE

The main idea behind deferred update technique is to defer or postpone any actual updates to a database until the transaction completed successfully and reaches its commit point. The changes made during the execution of the transaction are recorded in cache buffers and log but not in the database. After the transaction reaches it commit point and log is force written to disk, the updates are recorded in the database.

If a transaction fails before reaching its commit point there is not need to undo any operations because the transaction has not effected the database on disk. However, it may be necessary to redo the effect of operations of the committed transaction from the log because their effect may not yet have been recorded in the database in case of system crash.

This algorithm is also known as NO UNDO/REDO Recovery Algorithm because only Redo operation need to be performed and no undo operation is required. This algorithm is used when transactions are short and each transaction changes only few items in the-database. For long transactions there is potential of running out of buffer space because transaction changes must be held in cache buffers until the commit point.

2. IMMEDIATE UPDATE

In this technique when a transaction issues an update command, the database can be updated immediately without any need, to wait for the transaction to reach its commit point. However the changes are still recorded in the log (on disk) before they are applied to the database so that we can still make recovery in case of failure.

If the transaction fails after recording some changes in the database but before reaching its commit point, the effect o its operations on the database must be undone i.e. transaction must-be rolled back.

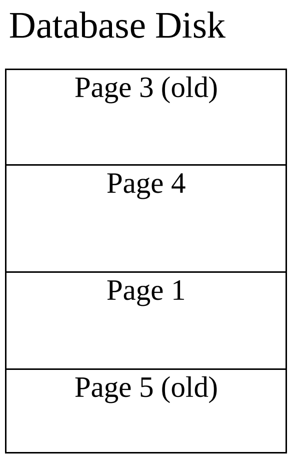
If all updates of a transaction are recorded in the database on disk before the transaction commits there is no need to Redo any operations of a committed transaction. So this recovery algorithm is known as UNDO/N0 REDO Algorithm. On the other hand, if all the updates are recorded in the database after the transaction commits then it is known as UNDO/REDO algorithm.

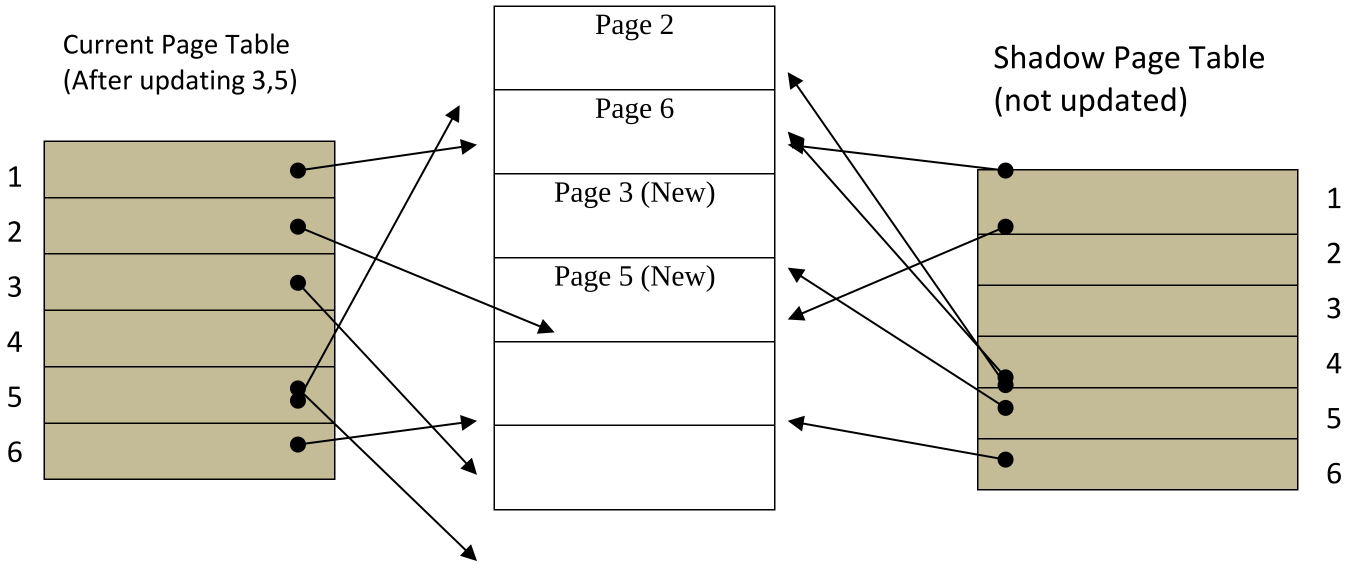
SHADOW PAGING

We have already studied the log based recovery techniques in which the log records are used to store changes made to the database which helps in recovering the database to consistent state in case of failures such as system crash etc. An alternative way to log based recovery to recover from the system crash and other failures is Shadow Paging. In this scheme the database is considered to be made up of logical units of storage called pages.

The pages are mapped into (of the same size as logical pages) Physical blocks of storage, by means of a page table with one entry for each logical page of the database . This entry contains the block number of physical storage where this page is stored. The shadow page scheme uses two page tables for a transaction that is going to modify the database.

The original page table is called the shadow page table and the transaction addresses the data base using another page table known as current page table. When the transaction starts both page tables are identical i.e. both page tables point to the same blocks of physical storage. The current page entries may change during the life of the transaction but shadow page table is never changed during the duration of the transaction. The changes are made to the current page when a transaction performs a write operation. All input and output operations use the current page table to locate database pages on disk. The shadow page table contains the entries that existed in the page table before the start of the transaction and point to blocks that were never changed by the transaction. The shadow page table is used for undoing the transaction because it remains unaltered by the transaction.





The shadow page technique is also known as Cut-of-Place updating

COMMIT OPERATION

The followings should be done in order to commit the transaction:

1. Ensure that all modifications made by the transaction which are still in buffers are propagated to the physical database (i.e. flush all modified pages in main memory to disk)
2. Output the current page table to the disk.
3. Output the disk address of the current page table to the fixed location in stable storage containing the address of shadow page table. This action overwrites the address of the old shadow page table. Therefore, the current page table has become the shadow page table and the transaction is committed.

When the transaction completes the current page table becomes the shadow page table.

FAILURE

In case of system crash during the transaction execution before it commits, it is sufficient to free the modified database pages and to discard the current page table. The state of the database before the execution of the transaction is available through the shadow page table and that state is recovered by reinstating the shadow page table.

Thus the database is returned to its state before the execution of transaction after the crash occurs and any modified pages are discarded. Thus, we noticed that there is no need to Undo or Redo the transaction as in case of log based recovery because by just modifying the pointers from shadow page table to the current page table we can make recovery since recovery involves neither Undo or Redo operations in the shadow paging scheme. So this technique can be characterized as no-Undo/ no-Redo technique for recovery.

If a system crash occurs after the last write operation then it will have no effect on the propagation of changes made by the transaction. These changes will be preserved and there is no need to perform Redo operation.

ADVANTAGES OF SHADOW PAGING TECHNIQUE

The following are the advantages of shadow paging technique:

1. Shadow Paging require fewer disk accesses than the log based method of recovery under certain circumstances.
2. Recovery from system crash using this scheme is relatively inexpensive and faster and is achieved without the overhead of logging.
3. No Undo and No Redo operations are required during the recovery using this technique.

DISADVANTAGES OF SHADOW PAGING TECHNIQUE

The following are the disadvantages of Shadow Paging Technique:

1. Since the updated database pages changes location on the disk so this makes it difficult to keep related database pages close together on disk without complex storage management strategies.
2. When a transaction commits, the original version of the changed blocks pointed to by the Shadow Page table have to be returned to the pool of free blocks otherwise such pages will I become inaccessible. So a "Garbage Collection" need to be performed periodically to reclaim such lost blocks.
3. The commit of a single transaction using Shadow Paging requires multiple blocks to be output such as actual data blocks, current page table and the disk address of the current page table. If a transaction contains large number of write operations then the overhead of writing data from main memory (RAM) to disk (Hard Disk) would be high, thus slowing the speed of executing transaction.
4. It is hard to extend Shadow p gang to allow multiple transactions to execute concurrently because it requires additional book keeping and in such a scheme some logging scheme is used as well.