Relational Calculus

① What to be retrieved. (NOT how to be retrieved)

B.Tech is a class → "is a class"(B.Tech) → Class(B.Tech).

Naresh is taller than Rajesh → Taller than(Naresh, Rajesh).

Based on Predicate

* Tuple oriented Relational Calculus.
* Domain " "

① Tuple oriented Relational Calculus

$\{t \mid Cond(t)\}$. t → tuple variable. Cond(t) → Cond. involving t.

eg: $\{t \mid Book(t) \text{ and } t.Price > 1000\}$ set of all the books whose price is greater than 1000.

To retrieve only some attributes.

$\{t.Title, t.Author, t.Price \mid Book(t) \text{ and } t.Price > 200\}$.

Range.

General Expression

$\{t_1.A_1, t_2.A_2 \ldots t_n.A_n \mid Cond(t_1, t_2, \ldots t_n, t_{n+1}, \ldots t_{n+m})\}$

where $t_1, t_2 \ldots t_n, t_{n+1} \ldots t_{n+m}$ are tuple variables, each A_i is an Attribute of the relation on which t_i ranges and Cond is a condition (formula) of tuple Relational Calculus.

To form a formula we use (and, or, not) and quantifiers like for all values (∀), or there exist (∃).

Bound variables: are those range variables when the meaning of the formula would remain unchanged if all the occurrence of range variable say 'x' were replaced by some other variable say 'y'.

ex: $x(x>3)$ means EXISTS $x(x>3)$

or EXISTS $y(y>3)$

Free variable: ---- formula changed.

ex: $x(x>3)$ and $x<0$ means

(1) Exists $x(x>3)$ and $x<0$.

Exists $y(y>3)$ and $x<0$ — equivalent to (1)

Exists $y(y>3)$ and $y<0$ — not equivalent to (1)

Domain Calculus: It varies in the type of variable used in formulas. Here variables range over single values from domains of attributes rather than ranging over tuples. To form a relation of degree 'n' for a query result, we must have 'n' of these domain variables - one for each attribute.

$\{x_1, x_2, \ldots x_n \mid \text{COND}(x_1, x_2, \ldots x_n, x_{n+1}, \ldots x_{n+m})\}$

Here $x_1 \ldots x_n \ldots x_{n+m}$ are domain variables that range over domain of attributes and COND is condition or formula.

Tuple Relational Calculus.

→ It is non procedural query language.

→ The expression/query in TRC is of form $\{t \mid P(t)\}$ where is P is a formula. Several tuple variables may appear in formula.

→ A tuple variable is said to free unless quantified by a ∃ or ∀

→ - - - - - - - bound if quantified by ∃ or ∀

Ex: ~~$\{t$~~ $\exists t \in r\,(Q(t))$

there exists a tuple t in relation r such that predicate Q(t) is true.

$\{t \mid \exists s \in loan\,(t[loan\text{-}no.] = s[loan\text{-}no.] \wedge s[amount] > 1200)\}$

The set of all the ~~ty~~ tuples t such that there exists a tuple s in relation loan for which values of t and s for the loan_no. attribute are equal, and value of s for the amount attribute is greater than 1200.

Symbols

∃ there exist. ∧ and

∈ belongs to ∨ or.

¬ not.

→ ~~form~~ TRC formula is built out of items

- $S \in r$
- $S[x] \ \Theta \ u[y]$ → Θ comparison operator $<, \leq, =, \neq, >, \geq$.
- $S[x] \ \Theta \ c$ → c is constant in domain of attribute x.

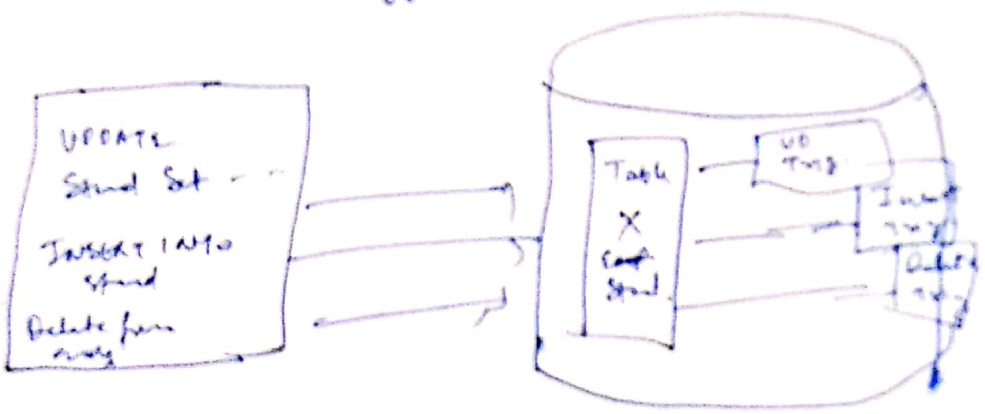
Rules

- An atom is formula.
- If P_1 is formula, then so are $\neg P_1$ and (P_1)
- & $P_1 \wedge P_2$ " " " " $P_1 \vee P_2$ $P_1 \wedge P_2$
- If $P_1(s)$ → s free tuple & r is relation that

$\exists s \in r(P_1(s))$ and $\forall s \notin r(P_1(s))$. are also formulas

TRIGGERS

- Stored procedure that is fired when INSERT, UPDATE, DELETE statement is issued against the table.
- it defines an action that DB should take when some DB related event occurs.
- Consist of PL/SQL & SQL statements to execute it as a unit and it can invoke other stored procedures.
- Code within Trigger - Trigger Body



Uses

- to derive colm values automatically
- enforce complex integrity constraints.
- " " Bus. rules
- customize complex security authorizations
- maintain Replicated Tables.
- audit data modifications

* Trigger do not accept parameters as procedures do.
- but automatically executed but procedures are explicitly invoked.

Parts of Trigger

① Triggering Event / Statement — Insert, Update, Delete

② Triggering Constraint (Optional)
- logical expression that must be True for trigger to fire.
- Conditionally control execution of Trigger.
- Specified by WHEN clause.

③ Trigger Action → PL/SQL code to be executed if trigg. condition statement is issued and restriction is true.

Types of Triggers

① Row level. ② Statement level ③ Before ④ After.

↳ for each row trigger executes once.

FOR EACH ROW CLAUSE (in Create TRIGGER command).

eg: Update to all except with Rno = 20. Then for each row checking has to be done. (This is handled by Trigger code when ever UPDATE is given).

Statement level → triggers only once for each transaction.

default type, CREATE Trigger Command.

eg:- Check on particular Day. Then trigger fires only once in System's date.

Triggers

Before & After (Events) - it must be specified when trigger is defined.

Before triggers are used when trigger action should determine execute trigger action before the triggering statement.

eg:- used to make a decision whether triggering statement should be allowed to complete? Thus with it we can eliminate unnecessary processing of triggering statement.

eg:- to prevent deletion on Sunday, ∴ use the statement level BEFORE trigger on DELETE statement

* are used to desire specific col. values before completing a triggering INSERT or UPDATE statement.

After trigger executes trigger action after triggering statement is executed.

eg:- Cascade delete operation.

INSTEAD of Trigger (oracle 8). - defined on the View
- what to do instead of performing the actions that invoked trigger.

Syntax for Creating a Trigger

CREATE [OR REPLACE] TRIGGER trigger_name ---- Trigg. Event

{BEFORE | AFTER}

{ | DELETE | OR} INSERT | [OR] UPDATE [OF Column_name,---] ;

ON table_name

[REFERENCING {OLD AS old, NEW AS new}]] Triggering restriction.

[FOR EACH ROW [WHEN condition]]

DECLARE :

Variable Declaration ;

Constant " ;

BEGIN

PL/SQL Sub. prgg. body ;

[Exception

Exception Body ;]

END ;

REFERENCING → CORRELATION Names.

Ques To Create a Trig. for emp table, which makes the entry in Ename column in Uppercase.

Ans:- CREATE or ---------- ---upper

BEFORE INSERT OR UPDATE of ename ON emp.

FOR EACH ROW

BEGIN

:new.ename := UPPER (:new.ename); (:old.ename);

END ;

Description: The above example also illustrates the use of the new keyword, which refers to the new value of the column, and the old keyword, which refers to the old values of the column. When referencing the new and old keywords in the PL/SQL block, they are preceded by colon(:)

Example 23.2: *To Create a trigger on the emp table, which shows the old values and new value of ename after any updation on ename of emp table.*

Solution:

```
CREATE OR REPLACE TRIGGER EMP_UPDATE
AFTER UPDATE OF ENAME ON EMP FOR EACH ROW
BEGIN
   DBMS_OUTPUT.PUT_LINE('OLD NAME ' || OLD.ENAME);
   DBMS_OUTPUT.PUT_LINE('NEW NAME' || NEW.ENAME);
END;
```

Example 23.3: *To Create a trigger on the emp table, which store the empno and operation in table auditor for each operation i.e. Insert, Update and Delete.*

Solution:

```
CREATE OR REPLACE TRIGGER EMP_AUDIT
AFTER INSERT OR UPDATE OR DELETE ON EMP
FOR EACH ROW
BEGIN
IF INSERTING THEN
    INSERT INTO AUDITOR VALUES(:NEW.EMPNO,'INSERT');
ELSIF UPDATING THEN
   INSERT INTO AUDITOR VALUES(:NEW.EMPNO,'UPDATE');
   ELSIF DELETING THEN
  INSERT INTO AUDITOR VALUES(:OLD.EMPNO,'DELETE');
END IF;
END;
```

Note: The same functionality can be achieved with the BEFORE type trigger also.

23.8 ERROR HANDLING IN TRIGGERS

The Oracle engine provides a procedure named *raise_appliation_error* that allows programmers to issue user-defined error messages.

Syntax:

```
RAISE_APPLICATION_ERROR (error_number, message);
```

Here:

error_number	It is a negative integer in the range −20000 to −20999
message	It is a character string up to 2048 bytes in length

This procedure terminates procedure execution, rolls back any effects of the procedure, and returns a user-specified error number and message. Following example makes use of the RAISE_APPLICAION_ERROR.

Example 23.4: ***To Create a trigger so that no operation can be performed on emp table on Sunday.***

Solution:

```
CREATE OR REPLACE TRIGGER EMP_SUNDAY
BEFORE INSERT OR UPDATE OR DELETE ON EMP
BEGIN
    IF RTRIM(UPPER(TO_CHAR(SYSDATE,'DAY')))
    ='SUNDAY' THEN
        RAISE_APPLICATION_ERROR(-20022,'NO OPERARTION CAN
BE PERFORMED ON SUNDAY');
    END IF;
END;
```

Example 23.5: ***To create a trigger, which verify that no record has the commission value greater than, salary of an employee in table emp.***

Solution:

```
CREATE OR REPLACE TRIGGER REC_CHECK
BEFORE INSERT OR UPDATE ON EMP FOR EACH ROW
BEGIN
    IF :NEW.SAL<:NEW.COMM THEN
        RAISE_APPLICATION_ERROR
        (-20000,'RECORD IS ILLEGAL');
    END IF;
END;
```

Example 23.6: ***To create a trigger, which verify that updated salary of employee must be greater than his/her previous salary.***

Solution:

```
CREATE OR REPLACE TRIGGER UPDATE_CHECK
BEFORE UPDATE ON EMP
FOR EACH ROW
BEGIN
    IF :NEW.SAL<:OLD.SAL THEN
        RAISE_APPLICATION_ERROR(-20000,'NEW SALARY
    CANNOT BE LESS THAN OLD SALARY');
    END IF;
END;
```

Example 23.7 : ***Create a trigger to implement the primary key constraint on column eno of table emptemp (eno, ename, job, sal, deptno).***

Solution:

```
CREATE OR REPLACE TRIGGER PRIMARY_KEY
BEFORE INSERT ON EMPTEMP
FOR EACH ROW
DECLARE
E EMTEMP.ENO%TYPE;
BEGIN
    IF (:NEW.ENO IS NULL) THEN
    RAISE_APPLICATION_ERROR(-20002, 'PRIMARY KEY
    VIOLATION BECAUSE IT CANNOT BE NULL');
END IF;
SELECT ENO INTO E FROM EMPTEMP WHERE ENO=:NEW.ENO;
RAISE_APPLICATION_ERROR(-20003, 'PRIMARY KEY VIOLATION
BECAUSE IT SHOULD BE UNIQUE');
EXCEPTION
WHEN NO_DATA_FOUND THEN
NULL;
END;
```

Explanation:

The trigger will be of row level before type having insert on table EMPTEMP as its firing statement, because the code to validate primary key need to fired before every insertion record into the table. Primary should be not null, so with the help of :NEW.ENO system can check that its value cannot be null in the insert statement. If it is null, the system will raise an application error and insert operation cannot be performed. The other constraint need to be fulfilled by the primary key is unique. It means that if :NEW.ENO is already present into the database, then insert operation cannot be performed and system should raise an application error. This is performed by the SELECT INTO statement which verifies the occurrence of :NEW.ENO value into the database. If record found for this value then application error is raised, and if it is not found then SELECT INTO statement raises NO_DATA_FOUND exception. This condition is handled by the exception hander with NULL statement, i.e., no operation will be executed and insert statement will be executed successfully.

If user executes the following insert statement, the trigger will be fired to restrict the execution of insert statement.

```
INSERT INTO EMPTEMP VALUES(NULL,'Ajay','Clerk',10000,10);
```

The following error message will appear:

```
INSERT INTO EMPTEMP VALUES(NULL,'Ajay','Clerk',10000,10);

ERROR at line 1:
ORA-20002: PRIMARY KEY VIOLATION BECAUSE IT CANNOT BE NULL
ORA-06512: at "PARTEEK.PRIMARY_KEY", line 5
ORA-04088: error during execution of trigger 'PARTEEK.PRIMARY_KEY'
```

If user tries to insert the duplicate eno into EMPTEMP with following insert table, the trigger will be fired to restrict the execution of insert statement.

```
INSERT INTO EMPTEMP VALUES(1,'Ajay','Clerk',10000,10);
```

The following error message will appear:

```
INSERT INTO EMPTEMP VALUES(1,'Ajay','Clerk',10000,10);

ERROR at line 1:
ORA-20003: PRIMARY KEY VIOLATION BECAUSE IT SHOULD BE UNIQUE
ORA-06512: at "PARTEEK.PRIMARY_KEY", line 8
ORA-04088: error during execution of trigger 'PARTEEK.PRIMARY_KEY'
```

Example 23.8: Create a trigger to implement the foreign key constraint on column deptno of table emptemp (empno, ename, job, sal, deptno) which refer to the deptno column of table depttemp (deptno, dname, loc).

Solution:

```
CREATE OR REPLACE TRIGGER FOREIGN_KEY
BEFORE INSERT ON EMPTEMP
FOR EACH ROW
DECLARE
DNO DEPTTEMP.DEPTNO %TYPE;
BEGIN
SELECT DEPTNO INTO DNO FROM DEPTTEMP WHERE
DEPTNO=:NEW.DEPTNO;
NULL;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE_APPLICATION_ERROR(-20004, 'FOREIGN KEY VIOLATED BECAUSE
 VALUE IS NOT FOUND IN THE PARENT TABLE');
END;
```

Explanation:

The trigger will be of row level before type having insert on table EMPTEMP as its firing statement, because the code to validate foreign key need to fired before every insertion of record into the table. The foreign key constraint on deptno column of EMPTEMP table with DEPTTEMP table as its parent table, requires that the value supplied for the deptno column in insert statement on EMPTEMP table should be present in the DEPTTEMP table. If it is not present into parent table DEPTTEMP then system should raise an application error and insert statement should not be executed. This task is performed by the SELECT INTO statement which verifies the occurrence of :NEW.DEPTNO value into dept table. If record found for this value then NULL statement is executed and no action is performed to restrict the insert statement. But, if the value is not found then SELECT INTO statement will raise

NO_DATA_FOUND exception and RAISE_APPLICATION_ERROR function is called to prevent the execution of insert statement.

Now, if user executes the following insert statements, the records will be inserted into the respective tables.

```
INSERT INTO DEPTTEMP VALUES(10,'COMPUTER');
INSERT INTO EMPTEMP VALUES(2,'Ajay','Clerk',10000,10);
```

If user violates the foreign key constraint by inserting a value of deptno that is not present in the master table DEPTTEMP, the system will raise the error as shown below.

```
INSERT INTO EMPTEMP VALUES(3,'Ajay','Clerk',10000,20);
ERROR RAISED:
INSERT INTO EMPTEMP VALUES(3,'Ajay','Clerk',10000,20);
                *

ERROR at line 1:
ORA-20004: FOREIGN KEY VIOLATED BECAUSE VALUE IS NOT FOUND
THE PARENT TABLE
ORA-06512: at "PARTEEK.FOREIGN_KEY", line 8
ORA-04088: error during execution of trigger 'PARTEEK.FOREIGN_KEY'
```

23.9 DICTIONARY INFORMATION ABOUT TRIGGERS

Oracle has provided us many dictionary views to show information about the triggers. These are DBA_TRIGGERS, USER_TRIGGERS and ALL_TRIGGERS. These views basically contain the same columns. Following is the list of columns of USER_TRIGGERS view with their descriptions.

TRIGGER_NAME

Gives the information of name of the trigger.

TRIGGER_TYPE

Gives the information about whether the trigger is a STATEMENT or ROW trigger.

TRIGGERING_EVENT

Gives the information about the event that causes the trigger to fire, which may be either INSERT,DELETE or UPDATE.

TABLE_NAME

The table to which the trigger is associated.

REFERENCING_NAMES

Gives the information of OLD or NEW referencing names you have chosen for the trigger. It contains NULL for STATEMENT trigger.

WHEN_CLAUSE

It contains the trigger restriction i.e. when clause which determines if the trigger will fire for that row. It also contain NULL for STATEMENT trigger.

STATUS

Tells about whether the trigger is ENABLED or DISABLED .

TRIGGER_BODY

Unlike stored procedures only the source code of database trigger is stored in the database. When we query this column of USER_TRIGGERS view it will return the PL/SQL code in the body of your trigger. Its data type is LONG. All these column are present in all the three types of views mentioned above. With the exception Of OWNER and TABLE_OWNER columns. These two appear only in the DBA_TRIGGERS and ALL_TRIGGERS views.

Lets say if somebody wants to know the Trigger Name, Trigger Type and Trigger Event of all the triggers associated with particular table , the following sql query will provide this information.

```
SQL > select trigger_name,trigger_type, triggering_event
      from USER_TRIGGERS
            Where TABLE_NAME ='ACCOUNTS';
```

The above query will give the list of trigger associated with ACCOUNTS table with the following attributes(before running the above stated query please make sure that you have created this trigger AUDIT_ACCOUNTS on ACCOUNTS table).

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT
AUDIT_ACCOUNTS	BEFORE EACH ROW	UPDATE OR DELETE

Suppose if we are interested in the detail of particular trigger. The following query can get it.

```
SQL> select trigger_type,triggering_event,table_name,status from
       USER_TRIGGERS where TRIGGER_NAME ='AUDIT_ACCOUNTS';
```

The above query will give the following detail of the AUDIT_ACCOUNTS trigger:

TRIGGER_TYPE TRIGGERING_EVENT TABLE_NAME STATUS

EFORE EACH ROW UPDATE OR DELETE ACCOUNTS ENABLED

Since the body of the trigger is also stored in USER_TRIGGERS view, so we can also see the body of trigger by the following query.

```
SQL > select TRIGGER_BODY from USER_TRIGGERS
        where TRIGGER_NAME ='AUDIT_ACCOUNTS';
```

23.10 ENABLING AND DISABLING TRIGGERS

When a trigger is created it is automatically enabled and is triggered whenever the triggering command and the execution command is true. An enabled trigger executes the trigger body if the triggering statement is issued. To disable the execution use the ALTER TRIGGER command with the DISABLE clause. A disable trigger does not execute the trigger body even if the triggering statement is issued.

We can disable / enable the trigger by the following syntax:

```
ALTER TRIGGER <trigger name> DISABLE / ENABLE
```

23.11 DROPPING TRIGGERS

Triggers may be dropped via the drop trigger command. In order to drop a trigger, you must either own the trigger or have the DROP ANY TRIGGER system privilege.

Syntax:

```
DROP TRIGGER trigger_name;
```

Example:

```
DROP TRIGGER emp_update;
```

FLASH BACK

A database trigger is a stored procedure that is fired when an INSERT, UPDATE, or DELETE statements is issued against the associate table. The name trigger is appropriate, as these are triggered (fired) whenever the above-mentioned commands are executed. A trigger defines an action the database should take when some database related event occurs.

A trigger is automatically executed without any action required by the user. A stored procedure on other hand needs to be explicitly invoked. This is the main difference between a trigger and a stored procedure.

Database triggers can be used for the following purposes: To derive column values automatically, to enforce complex integrity constraints, to enforce complex business rules, to customize complex security authorizations, to maintain replicate tables and to audit data modifications.

A database trigger has three parts:

- ✦ Triggering event or Statement.
- ✦ Trigger constraint (Optinal)
- ✦ Trigger action

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 Relations 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, homogeneous 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 [illegible] 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 in 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 atributes 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 ⟶ S_Name

Rollno ⟶ S_phone

Rollno ⟶ Course

Rollno ⟶ 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

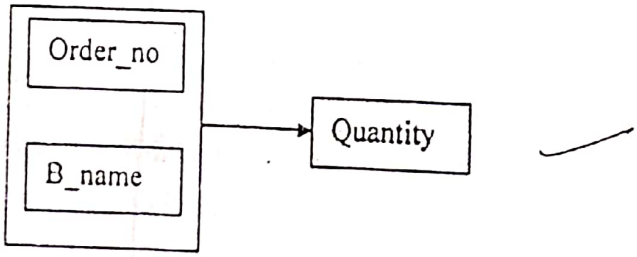
ORDER_BOOK Relation

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	[illegible]	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

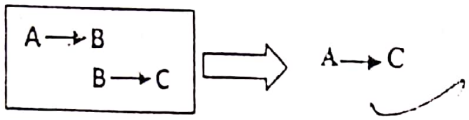
Before discussing the third Normal Form (3NF) we should be aware of the transitive dependency. Suppose there exists a Relation R with the attributes A, B and C. A is the primary key and B and C are the non key attributes.

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)



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.

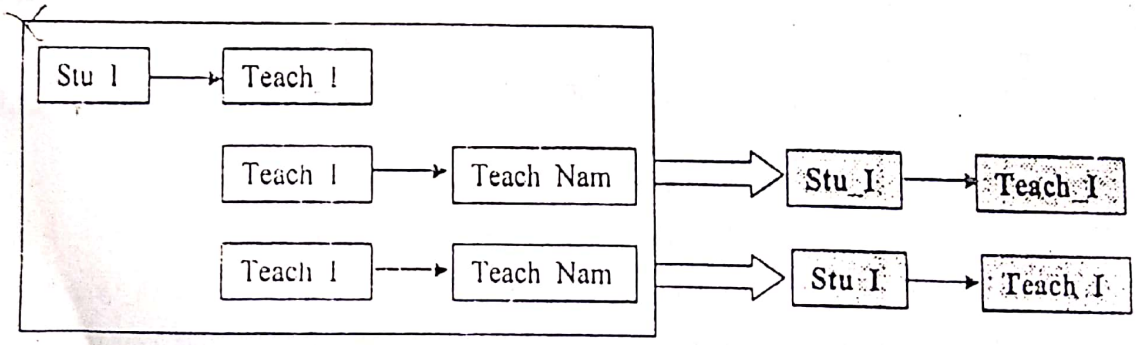
In the STUDENT Relation FIG the following functional dependencies hold.

Stu_Id →Stu_Name Stu_Id →Teach_Id

Teach_Id →Teach_Name Teach_Id →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

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 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	5558	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

GRAD_COU

Stu_Id	Course	Grade

Rajesh	5708
Kapil	5901
Sunil	5658
Ankit	5816

5705	Computer	A
5801	Computer	C
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 each pair of (Empname, Language) values in the in the EMPLOUEE Relation, the associated set of **Equipment** values is determined only be **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 ⟶ **Equipment**

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

Empname ⟶ **Language**

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

EMPLOYEE

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 authentication can 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 the stored 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 to the 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 da: 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

- Subjects

  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.

- Objects

  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.

- Access Rights

  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.

THE AUTHORISED 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**

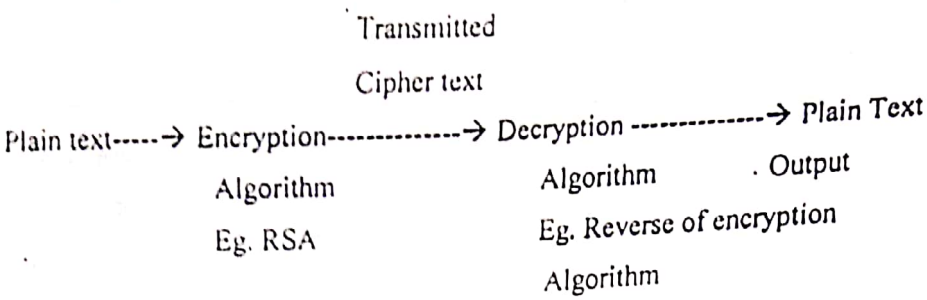
Key known only to individual user and used to decrypt data need not be transmitted

to the site doing encryption.

4) Cipher text

5) Decryption algorithm

Following diagram showing public key encryption:



Public key encryption 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 he 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:-

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) Problem of dirty reads

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) Unrepeatable reads

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.

Schedule

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

Types of schedules:-

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.

Locks

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 to use 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	
Bx=Bx-50	Read By
Update Bx	By=By-60
Unlock Bx	Update By
Read By	Unlock By
By=By+50	Read Bz
Update By	Bz=Bz+60
Unlock By	Update Bz
	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:

Shared Lock

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.

Exclusive Lock

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 of the balances in account A & B and other transaction transfers a Rs 200 from account A to account B.

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)
T :Unlock(B)
T :Lock S(B)
T :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 schedu e has occurred even th ugh both the transactions locked and unlocked the data items used by the n.

Two- Phase Locking(2PL)

Locking may help in achieving serializibility but it does not always guarantee it. Or : way of guarantee serializability is to use, an additional protocol concerning with the po itioning 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 oj erations precede the first unlock operation in the transaction. Each transac on in 2PL is sues lock and unlock request in two phases:

1 Growing phase
2 Shrinking phase

Gro ving phase

The growing phase is also known as expanding or first phase. In this phase the tra action ac ires all locks needed but cannot release any locks.

Sh inking p se