UNIT-III

Queries, Constraints, Triggers: The Form of Basic SQL Query, Union, Intersect, and Except, Nested Queries, Aggregate Operators, Null Values, Complex Integrity Constraints in SQL, Triggers and Active Database.

The basic form of an SQL query:

SELECT * DISTINCT+,*| column_name1, column_name2…) FROM table_name

WHERE condition + *GROUP BY column_list+ *HAVING condition+ *ORDER BY column_list.

- SELECT specifies which columns are to appear in the output DISTINCT eliminates duplicate
- FROM specifies the tables to be used
- WHERE filters the rows according to the condition The where condition is a boolean combination (using AND, OR, and NOT) of conditions of the form expression op expression where op is one of the comparison operators (<=, =, <>, >=, >)
- GROUP BY forms groups of rows with the same column value
- HAVING filters the group
- ORDER BY sorts the order of the output

Set Operations:

❖ Union,
❖ Except(minus)
❖ Intersect

sname	Account
Aijay	A1
Vijay	A2
Ram	A3

student1

sname	loan
Vishal	L1
Ram	L2

student2

Union (U):- it is the binary operation between the two relations r and s. denoted by r U s. It is the union of set of tuples of the two relations. Duplicate tuples are automatically removed from the result. A tuple will appear in r U s if it exists in r or in s or both for U to be possible, r and s must be compatible.

a) r and s must be of same degree i.e. they must have same no of attributes

b) For all i, the domain of i th attribute of r must be same as the domain of the i th attribute of s.

Query:-. Get the names of those students who have either account or loan or both at the bank

SQL: select sname from student1 union select sname from student2;

Result:

Sname
Aijay
Vijay
Ram
Vishal

Except (-): The set difference operation (r -s) between two relations r and s produced a relation with tuples which are in r but not there in s. To possible r-s, r and s must be compatible

Cardinality of r-s = cardinality (r) – cardinality (r ∩ s)

Query:. Get the names of those students who have account in the bank but do not have loan

SQL: select sname from student1 minus select sname from student2;

Result:

Sname
Ajay
Vijay

Intersect (∩): his operation r ∩ s between the relations r and s produced a relation with tuples which are there in r as well as s. For is to be possible, relations r and s must be compatible

Query: get the names of those students who have account as well as loan

SQL: select sname from student1 intersect select sname from student2;

Result:

Sname
Aijay
Vijay

Data types:-

- Each value in oracle is maintained by a data type.
- The value of one data type is different from other data type.
- The data type defines the domain of values that each column can contain

Character data types:-

This data type is used to store character data. Different character data types are

1.char

2.varchar2

1. **Char data type: -** it specifies fixed length character string. Size should be specified. If the data is less then original specific size, blank spaces are applied. The default length is 1byte and maximum length is 200 bytes.

   Ex: - char (10);
2. **Varchar2 data types: -** it specifies the variable length character string. It occupies only that space for which the data is supplied. The maximum size is 1byte and the maximum size is 400 bytes.

   Ex: - varchar2 (10);
3. **Number data types:-**
   - a) **number(P, S)**
     - i. P is precision, range is 1 to 38
     - ii. S is scale; range is -84 to 12
     - iii. Ex: - number (8, 3);
   - b) **Float:** - it is used to specify floating point values. It specifies decimal precision 38.
   - c) **Long data types: -** these are used to store very large text strings. A single table can have only one long column.
4. **Date and time data type:-**
   - **Date:** - it is used to store date information. The default date format in oracle is DD-MM-YYY Ex:- 29-07-2019
   - **Time:** this is used to store time information. It has atleast 8 positions embedded in single quotes. „HH:MM:SS"

     Ex: - 11:07:05
   - **Time stamp:** - it includes both time and date along with minimum 6digits representing decimal fraction of seconds. The format is „DD-MM-YYYY HH:MM: SS"

     Ex: - "31-05-1950 01:02:05 123456"

5. Large object data types: - these can store large and unstructural data like text, image, video and special data. the max size is up to 4 GB

   The types are
   - BLOB(binary large object)
   - CLOB(character large object) Maximum size is 4 GB

6. Raw and long raw data types: - these are used to store binary data or byte strings. These are variable length data types. They are mostly used to store graphics, sound documents etc.

SQL:

Structured Query Language is used to perform operations on the records stored in the database such as updating records, deleting records, creating and modifying tables, views, etc.

SQL is just a query language; it is not a database. To perform SQL queries, you need to install any database, for example, Oracle, MySQL, MongoDB, PostGre SQL, SQL Server, DB2, etc.

Types of SQL commands:-

1. Data definition language (DDL):- it is used to define the database schema.

   The commands used under this languages are:-
   1. create
   2. after
   3. drop

Syntaxes and examples: -

1. **create:-** Used to create table structure into a database
   Syntax:-
   Create table <table-name>(col1 datatype[size] constraints list, col2 datatype[size] constraints list,--------------);

   Ex: - create table student (sid number (4)primary key, sname varchar2(10)not null);

2. **alter:-** used to alter the table definition

   a) alter with add option:- syntax:-

   alter table <table-name> add <col-name> datatype[size]

   ex:- alter table dept add loc1 varchar2(10);

   b) alter with drop option:-

   syntax:- alter table <table-name>drop column <col-name>;

   ex:- alter table dept drop column loc1;

c) alter with modify option:-

syntax:- alter table <table-name>modify <col-name>datatype[size];

ex:- alter table dept modify loc varchar2(10);

d) alter with rename option:-

syntax:- alter table<table-name> rename column<old col name> to <new column>

ex:- alter table rename column loc to location

3. **drop:**- used to drop a database table permanently.

   Syntax:- drop table<table-name>

   Ex:- drop table dept;

2. Data manipulation language(DML):-

These are used to manipulate the data in the databases. The commands used in the language are

1. Insert
2. Update
3. Delete

Syntaxes and examples:-

1. **Insert:**- used to insert rows into a table

   Syntax:- insert into <table-name>(col1,col2,---,coln)values(val1,val2,----,valn);

   Ex:- insert into dept(deptno, dname,loc)values(50,"xyz","hyd");

2. **Update:**- used to update rows of table

   Syntax:-update <table-name>set <column-name>=<value>where <col-name>=<value>

   Ex:- update dept set dname="pqr" where deptno=50;

3. **Delete:-** used to delete rows from a table

   Syntax:- delete from <table-name> where <col-name>=<value>;

   Ex:- delete from dept where deptno=50;

3. Data query language (DQL):- It is used to extract data from database tables.

The command comes under the language is

1. **Select**

Syntax:- Select <col-list>, <group functions>from <table-name> where <condition> groupby <column>having<group condition>orderby<column-name>

Ex:- Select deptno, sum(sal), max(sal), min(sal), avg(sal) from emp Where job="clerk" group by deptno having avg(sal)>1000 order by deptno;

4. Data control languages: - These commands control the user access to the database.

The commands comes under these languages are

1. Grant
2. Revoke

Grant: - used to grant the permissions to the user on the db tables. Syntax: - grant <priviliges-name>ON <object name>to<user-name>

Ex: - grant select, insert, delete on emp to operators;

Revoke: - used to take back the permissions from the user.

Syntax: - revoke<priviliges-name>ON <object name>from<user-name>

Ex: - revoke insert, delete on emp from operators;

5. Data administrative language(DAL):- These commands are used for audit, the

commands are

1. Start audit;
2. Sleep audit;

6. Transaction control language(TCL):-These commands are used to control the transactions

1. Commit
2. Rollback
3. Savepoint

Syntaxes:-

1. commit;
2. rollback;
3. rollback to<save point name>;

Relational set operators:-

1. **union:-** merges the output of two or more queries into a single set of rows and columns.

   Ex:-select job from emp where deptno=10 union select job from emp where deptno=30;

2. **union all:-** union suppresses the duplicates where as union all will also display duplicates.

   Ex:- select empno, ename from emp where deptno=10 union all select empno, ename from emp where deptno=30**;**

3. **intersect:-** this operator returns the common rows that are common between two queries.
   Ex:- select job from emp where deptno=20 intersect select job from emp where deptno=30;

4. **minus:-** this returns the rows unique to the first query.
   Ex:- select job from emp where deptno=20 minus select job from emp where deptno=10;

Sub queries/Nested queries/ sub select/inner select:-

- It is the concept of placing one query inside the other query
- Inner or sub query returns a value which is used by the outer query.

Types of subqueries:-

1. Single row sub query
2. Multiple row subquery
3. Multiple column subquery
4. Inline subquery
5. Correlated subquery

1) **Single row subquery**: - These returns only one row from inner select statement.
   It uses only single row operator. (>,=,<,<=,>=)
   Ex: - select ename, sal, job from emp where sal>(select sal from emp where empno=7566);
2) **Multiple row subquery: -** The subqueries return more than one row are called multiple row sub queries. In this case multiple row operators are used.

   a) IN equal to any number of list.
   b) ANY compares value to each returned by subquery

      .
      I. <any less than the max value
      II. >any greater than min value

   c) ALL compares value the each value returned by subquery

      <any less than the max value

      >any greater than min value

   Ex: - select empno, ename, job, from emp where sal<any(select sal from emp whee job=‟clerk‟);

3) **Multiple column subquery**:- in this the subquery return multiple columns.

   Ex:- select ename, deptno from emp where(empno.deptno)in(select empno, deptno from emp where sal>1200);

4) **Inline subquery:-**in this the subquery may be applied in select list and inform clause.
   Ex:- Select ename, sal, deptno from (select ename, sal, deptno, mgr, hiredate from emp);

5) **correlated subquery:-** in this the information of outer select participate as a condition in inner select.

   Ex:- select deptno, ename, sal, from emp x where sal>(select avg(sal)from emp where x.deptno=deptno)orer by deptno;

   - here first outer query is executed and it pass the value of deptno to the inner query then the inner query executed and give the result to the outer query.

Aggregate functions: -

These are used to display the aggregated data from group of values.

1. **max():-** used to get max value from the list of values

   ex:- select max(sal) from emp;

   output:- MAX(sal)

   10000

2. **min():-** used to get min value from the group of values.

   Ex: select min(sal) from emp;

   Output:- MIN(sal)

   800

3. **sum():-** used to get the total sum of values.

   ex:- select sum(sal)from emp;

   output:- SUM(sal)

   37525

4. **avg ():-** used to get the average value of the given values.

   Ex:- select avg(sal) from emp;

```
Output:- AVG(sal)

---------------

2680.35714
```

5. **count():-** used to count the list of values

ex:- selsct count(sal)from emp;

```
output:- COUNT(sal)

----------------

    14
```

6. **Order by clause:-** it is used sort the values of column in ascending or descending oreder.

Ex:- select ename from emp order by ename;

Output:-

```
ENAME

------------

Adems Allen Blake Clerk Ford James Jhons King Martin Miller Scort Smith Turner
                                                                        Ward

8  rows are selected
```

Ex:- select sal from emp order by sal desc;

Output:-

```
SAL

-------

10000

5000

3000

3000

2975

2850

2450

7 rows selected
```

By default order by clause sort the values in ascending order

Group by clause:-this is used to display the group wise data i.e. department, job wise

Select deptno, count(*) from emp group by deptno;

Output:-

```
DEPTNO                          COUNT (*)
-------------                   --------------
  30                            6
  20                            5
  10                            3
```

Having clause:

It is used to define conditions on a grouping column. Where clause defines conditions on the selected columns where has the having clause places conditions on groups created by the group by clause.

Ex: - select deptno, min(sal) from emp group by deptno having min(sal)>800;

```
DEPTNO                                  MIN (sal)
-------------                           --------------
30                                      950
10                                      1300
```

Ex:- select job, min(sal)from emp group by job having min(sal)>800;

Output:-

```
JOB                                     MIN (SAL)
---------                               ----------------
Salesman                                1250
President                               5000
Manager                                 2450
analyst                                 3000
```

ex:- select job, sum(sal), avg(sal), min(sal), max(sal) from emp where deptno=20 group by job having avg(sal)>1000 order by job;

```
JOB                 SUM (SAL)           AVG (SAL)    MIN(SAL)     MAX(SAL)
------------------------------------------------------------------------------
Analyst               6000                3000        3000           3000
Manager               2976                2975        2975           2975
```

Importance of null values:-

- A 'NULL' is a term used to represent a missing value.
- Null is undefined, unknown, and unavailable and it is not equal to zero or a space.
- The regular operators like +, -, *, %, =, <, >, <=, >= will be fail with null values.

Why should we avoid placing of null values into DB:-

- All arithmetic and comparison operators will fail with null values i.e. if we add a column to the null value column then the result will become null only.
- A null will occupy large space in a database.
- We use two operators with the null values.
    1. is null
    2. is not null

Ex:- select ename from emp where column is null

We have the following functions to handle with the null values.

1. nvl()
2. nvl2()
3. coalesce()

nvl(expr/column, default value):-

This function returns first argument value if the first argument is not null, if it is null then it return the 2nd argument value.

Ex: - select nvl(column,0) from emp;

In the output of above query if column is null then the default value (2nd argument) i.e. 0 will be displayed, if column is not null then that value is displayed as it is.

nvl2 (expr1/column1, expr2/column2, expr3/column2):-

if the first argument value is not null then this function returns the value of 2nd argument, if the first argument value is null thoutpien this function returns the value of 3rd argument.

ex:- select nvl2(comm, sal+com,sal) from emp;

output:-

```
nvl2(comm, sal+com, sal)
--------------------------------
```

```
800
1900
1750
2975
2650
2850
2450
7 rows selected.
```

Coalesce (expr1/column1, expr2/column2, --------- expr n/column n):-

It takes 'n' arguments. This function accepts two or more arguments and returns the first not null value in the list. If all the arguments contain null values then this function returns a null value.

Ex: - select coalesce (20, 30, null) from dual;

Output:-

20 first not null value.

Select coalesce (null, null, 30) from dual;

Output:-

30 first not null value in the argument list.

Joins: - joins is a query that combines rows from two or more tables or views

- if some column name appears more than one table, the name must be prefixed with table name.
- To join n tables together, we need a minimum of n-1 conditions.

Join types:-

1. Simple join/equi join/inner join
2. Non equi join
3. Self join
4. Cartesian product
5. Natural join
6. Outer join

1. **Simple join:-** in this the join condition containing equality operator.

   Ex:- select E.empno, E.ename, D.deptno, D.dname, from emp E, dept D

   where $\underbrace{\text{E.deptno=D.deptno}}_{\text{Join condition}}$;

2. **Non equi join:-** in this no column of one table will not corresponds to any column of other table means the domain of no column in a table is not same as the domain of other table.
   - ➢ in this type no equal operator based on common columns in the join condition.

   Ex: - select E.ename, E.sal, S.grade from emp E, salgrade S where E.sal between S.losal and S.hisal;

3. **self join:-** it is a join of table itself.

   Ex:- select E1.ename “employee name”, E2.anme “managers name”,

   From emp E1, emp E2 where E1.mgr=E2.empno;

4. Cartesian product:- the Cartesian product is a join without a join condition consider the following relations

   student1

sname	account
Ajay Vijay ram	A1 A2 A3

   Student2

Sname	loan
Vishal ram	L1 L2

   Student1 * stuednt2

Student1.sname	account	Student2.sname	loan
Ajay	A1	Vishal	L1
Ajay	A1	Ram	L2
Vijay	A2	Vishal	L1
Vijay	A2	Ram	L2
Ram	A3	Vishal	L1
ram	A3	ram	L2

   SQL ex:- select * from student1,student2;

5. Natural join:- natural join is equal to the following sequence of operators
   a) Cartesian product of two relations
   b) Select the tuples based on the common column(attributes) of the two relations.
   Removing the duplicate attributes from the resultant relation natural join of

   student1*student2

sname	account	Loan
ram	A3	L2

   SQL ex:- select * from student1 natural join student2;

6. Outer join:- outer join extends the result of natural join. natural join will give the tuples only based on the common attributes of the two relations. The information of the other tuples will not be given by the natural join. it is possible to get such tuples information by using outer join.

There are three types of outer joins:-

1. Left outer join(L.O.J)
2. Right outer join(R.O.J)
3. Full outer join(F.O.J)

L.O.J:- it gives the full information of left side table (1st table)along with the natural join.

sname	account	sname	loan
Ram Ajay vijay	A3 A1 A2	Ram Ajay ajay	L2 Null Null

R.O.J:- it gives the full information about right side table (2nd) along with the natural join.

sname	account	sname	loan
Ram	A3	Ram	L2
vishal	Null	vishal	L1

R.O.J:- it will give the full information about lest and right side tables along with natural join.

sname	account	sname	loan
Ram	A3	Ram	L2
Ajay	A1	Ajay	Null
Vijay	A2	Vijay	Null
vishal	null	vishal	L1

SQL Queries:-

- Select * from student1 left outer join students on

  stydent1.sname = student2.sname;

- Select * from student1 right outer join students on

  stydent1.sname=student2.sname;

- Select * from student1 full outer join students on

  stydent1.sname=student2.sname;

Complex integrityconstraints:

We have discussed the integrity constraints in the unit-II but we can make them more complex by defining a table with two or more foreign keys in a table by referring primary keys of different tables as shown below

```
SQL> create table sailors(sid number(2)primary key,sname varchar2(10),rating num
        ber(2),age float);

Table created. SQL> desc sailors;

Name        Null?                         Type
----------------------------------------- -------- ----------------------------
SID        NOT NULL                       NUMBER(2)
SNAME                                     VARCHAR2(10)
RATING                                    NUMBER(2)
AGE                                       FLOAT(126)

SQL> create table boats(bid number(3)primary key,bname varchar2(10),color
       varcha r2(10));

Table created.

SQL> desc boats;

Name        Null?                         Type
----------------------------------------- -------- ----------------------------
BID        NOT NULL                        NUMBER(3)
BNAME                                     VARCHAR2(10)
COLOR                                     VARCHAR2(10)

SQL> create table reserves(sid number(2) references sailors(sid),bid number(3)re ferences
                                                   boats(bid),day date);

Table created. SQL> desc reserves;

Name   Null?                              Type
----------------------------------------- -------- ----------------------------
SID                                       NUMBER(2)
BID                                       NUMBER(3)
DAY                                       DATE
```

Sid and bid in the above table are foreign keys which are referring from the tables sailors and boats.

PL/SQL

Basic Syntax of PL/SQL which is a block-structured language, this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts

S.No	Sections & Description
1	**Declarations** This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Following is the basic structure of a PL/SQL

```
DECLARE
<declarations section>
```

```
BEGIN
<executable command(s)> EXCEPTION
<exception handling> END;
```

The 'Hello World' Example

The end; line signals the end of the PL/SQL block. To run the code from the SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code. When the above code is executed at the SQL prompt, it produces the following result

```
Hello World
```

The PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, identifiers are not case-sensitive. So you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

```
DECLARE
message varchar2(20):= 'Hello, World!';
 BEGIN
dbms_output.put_line(message);
END;
/
```

The PL/SQL Delimiters

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL –

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
;	Statement terminator
:=	Assignment operator
=>	Association operator

||	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
- -	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

```
DECLARE
-- variable declaration
message varchar2(20):= 'Hello, World!';
BEGIN
/*
* PL/SQL executable statement(s)
*/ dbms_output.put_line(message);
END;
/
```

The PL/SQL Comments

Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler. The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

When the above code is executed at the SQL prompt, it produces the following result –

```
Hello World
PL/SQL procedure successfully completed.
```

PL/SQL Program Units

- PL/SQL block
- Function
- Package
- Package body
- Procedure
- Trigger
- Type
- Type body

Triggers:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match. Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATES, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF }
{INSERT | UPDATE  | DELETE} [OF col_name] ON table_name
[REFERENCING OLD AS o NEW AS n] [FOR EACH ROW] WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Types of Triggers in Oracle

Triggers can be classified based on the following parameters.

- **Classification based on the timing**
    - BEFORE Trigger: It fires before the specified event has occurred.
    - AFTER Trigger: It fires after the specified event has occurred.

- INSTEAD OF Trigger: "INSTEAD OF trigger" is the special type of trigger. It is used only in DML triggers. It is used when any DML event is going to occur on the complex view.

- **Classification based on the level**
  - STATEMENT level Trigger: It fires one time for the specified event statement.
  - ROW level Trigger: It fires for each record that got affected in the specified event. (only for DML)
- **Classification based on the Event**
  - DML Trigger: It fires when the DML event is specified (INSERT/UPDATE/DELETE)
  - DDL Trigger: It fires when the DDL event is specified (CREATE/ALTER)
  - DATABASE Trigger: It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)

: NEW and: OLD Clause

In a row level trigger, the trigger fires for each related row. And sometimes it is required to know the value before and after the DML statement.

Oracle has provided two clauses in the RECORD-level trigger to hold these values. We can use these clauses to refer to the old and new values inside the trigger body.

- **:NEW** – It holds a new value for the columns of the base table/view during the trigger execution
- **:OLD** – It holds old value of the columns of the base table/view during the trigger execution

INSERT	UPDATE	DELETE
: NEW	VALID	VALID
: OLD	INVALID	

Examples onTriggers: **client_master**

```
NO       NAME   BAL_DUE   ADDRESS        CITY

---------- ---------- ----------      ----------      ----------
 1          abc        300          sacet        vetapalem
 2          xyz        500          saec          chirala
 3          pqr        700          sacet         vetapalem
```

audit_client

```
SQL> create table audit_client(clientno number,name varchar2(10),bal_due
        number(10,2),operation varchar2(10),userid varchar2(10),odate date);
```

Creating BEFORE UPDATE Trigger:

```
create or replace trigger audit_trail before update on client_master for each row
declare oper varchar2(10);
clientno client_master.clientno % type;
name client_master.name % type;
bal_due client_master.bal_due%type;
 begin
if updating then oper:='update';
 end if;
if deleting then
oper:='delete';
end if;
clientno:=:old.clientno;
name:=:old.name;
 bal_due:=:old.bal_due;
insert into audit_client values(clientno,name,bal_due,oper,user,sysdate);
 end;
/
Trigger created.
SQL> select * from client_master;
NO       NAME      BAL_DUE      ADDRESS      CITY
---------- ----------        ----------        ----------        ----------
 1          abc            300         mpes        Guntur
 2          xyz            500         mpes        Guntur
 3          pqr            700         mpes        Guntur

SQL> update client_master set bal_due=bal_due+1002 where clientno=2;
1 row updated.
```

```
SQL> select *from client_master;
NO        NAME      BAL_DUE      ADDRESS      CITY
---------- ----------        ----------       ----------        ----------
 1           abc                 300          mpes         Guntur
 2           xyz                 500          mpes         Guntur
 3           pqr                 700          mpes         Guntur
SQL> select *from audit_client;

NO      NAME    BAL_DUE   OPERATION  USERID         ODATE
---------- ----------    ----------       ----------          ----------         ---------
2           xyz          500            update           SCOTT       29-JUL-19
```

Creating AFTER DELETE Trigger:

```
SQL> create or replace trigger audit_trail after delete on client_master
for each row declare oper varchar2(10);
clientno client_master.clientno%type;
 name   client_master.name%type;
bal_due client_master.bal_due%type;
begin
if updating then oper:='update';
end if;
if deleting then oper:='delete';
end if;
clientno:=:old.clientno;
name:=:old.name;
 bal_due:=:old.bal_due;
insert into audit_client values(clientno,name,bal_due,oper,user,sysdate);
end;
/
Trigger created.
```

```
SQL> delete from client_master where clientno=3;
1 row deleted.
SQL> select *from client_master;

NO        NAME      BAL_DUE      ADDRESS     CITY
---------- ----------       ----------       ----------       ----------
 1          abc             300         mpes        Guntur
 2          xyz             500         mpes        Guntur

NO        NAME    BAL_DUE OPERATION  USERID        DATE
---------- ----------      ----------         ----------       ----------        ---------
      2       xyz        500            update      SCOTT       29-JUL-19
      3       pqr        700            delete      SCOTT       29-JUL-19
```

Creating INSTEAD OF Trigger:

```
create or replace trigger instead_of_view instead of update on client_master_view for each
  row
begin
update audit_client set name=:new.name where clientno=:old.clientno;
end;
/
Trigger created.
SQL> select *from client_master_view;
NO        NAME      BAL_DUE      ADDRESS     CITY
---------- ----------       ----------       ----------       ----------
 1          abc             300         mpes        Guntur
 2          xyz             500         mpes        Guntur


SQL> select *from audit_client;

NO        NAME    BAL_DUE OPERATION  USERID        DATE
---------- ----------      ----------         ----------       ----------        ---------
      2       xyz        500            update      SCOTT       29-JUL-19
      3       pqr        700            delete      SCOTT       29-JUL-19
```

```
SQL> update client_master_view set name='ffff' where clientno=2; 1 row updated.

SQL> select *from client_master_view;

NO       NAME      BAL_DUE       ADDRESS      CITY

---------- ----------        ----------       ----------       ----------
 1          abc                300         mpes        Guntur
 2          xyz                500         mpes        Guntur
       SQL> select *from audit_client;

NO       NAME    BAL_DUE OPERATION  USERID        DATE

---------- ----------      ----------        ----------      ----------       ---------
      2       xyz         500            update       SCOTT       29-JUL-19
      3       pqr        700            delete       SCOTT       29-JUL-19
```

Try to create a trigger using FOR EACH STATEMENT (not in oracle)

```
create or replace trigger for_each_statement after insert or update or delete on client_master
for each statement
begin
delete from aa;
 end;
/
```

It will give the following error:

```
for each statement
* ERROR at line 3:
ORA-01912: ROW keyword expected
```

If we use ROW in place of STATEMENT then

```
create or replace trigger for_each_statement after insert or update or delete on client_master
for each row
begin
delete from aa;
6*
end;
SQL> /
Trigger created.
```

```
SQL> select *from aa;
X             Y
----------    ----------
12            jjjjjj

SQL> select *from client_master;

NO      NAME      BAL_DUE      ADDRESS      CITY
---------- ----------      ----------      ----------      ----------
 1        abc          300        mpes       Guntur
 2        xyz          500        mpes       Guntur
SQL> update client_master set name='hhhh' where clientno=1;
1 row updated.
SQL> select *from client_master;
NO      NAME      BAL_DUE      ADDRESS      CITY
---------- ----------      ----------      ----------      ----------
 1        abc          300        mpes       Guntur
 2        xyz          500        mpes       Guntur
no rows selected
```

Active Database:

A database that has the ability to spontaneously react to events occurring inside as well as outside the system is called active database. The ability to respond to external events is called active behaviour. The active behaviour is based on the rules that integrate a event with the desired effect. This behaviour is commonly defined in terms of ECA rules allowing system to react to specific events.

Active Rules:

- The active behavior is achieved through theθ production rules/ active rules.
- The active rules are stored programs called triggers that are fired when an event occurs.
- Triggers are written to respond to DML(select,θ insert etc), DDL(create, alter etc) and Database Operations(Log-On, Log-Off)
- These triggers can be defined on table/view orθ the database to which event is associated**.**

UNIT-3
SCHEMA REFINEMENT AND NORMALISATION

Unit 3 contents at a glance:

1. Introduction to schema refinement,
2. functional dependencies,
3. reasoning about FDs.
4. Normal forms: 1NF, 2NF, 3NF, BCNF,
5. properties of decompositions,
6. normalization,
7. schema refinement in database design(Refer Text Book),
8. other kinds of dependencies: 4NF, 5NF, DKNF
9. Case Studies(Refer text book)

.

1. Schema Refinement:

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is **decomposition.**

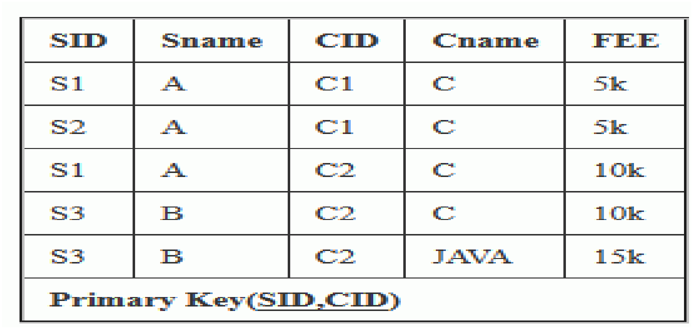
Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

Anomalies or problems facing without normalization(problems due to redundancy) :

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

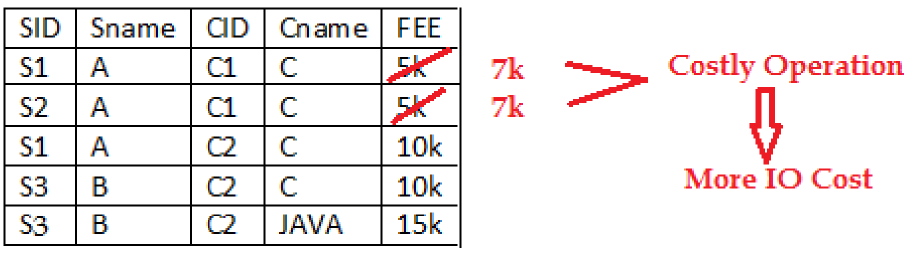
1.insertion anomalies : It may not be possible to store some information unless some other information is stored as well.

2.redundant storage: some information is stored repeatedly

3.update anomalies: If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.

4.deletion anomalies: It may not be possible to delete some information without losing some other information as well.

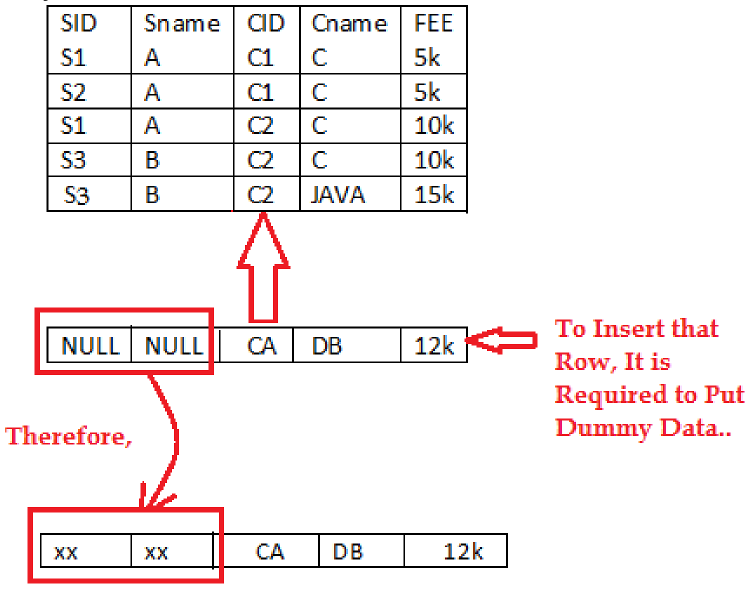
Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.



SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~
S2	A	C1	C	~~5k~~
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

InsertionAnomalies: New course is introduced C4, But no student is there who is having C4 subject.



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

NULL	NULL	CA	DB	12k

xx	xx	CA	DB	12k

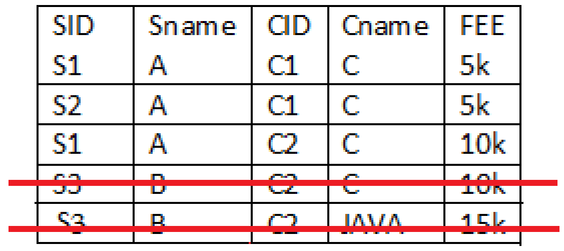
To Insert that Row, It is Required to Put Dummy Data..
Therefore,

Because of insertion of some data, It is forced to insert some other dummy data.

Deletion Anomaly :

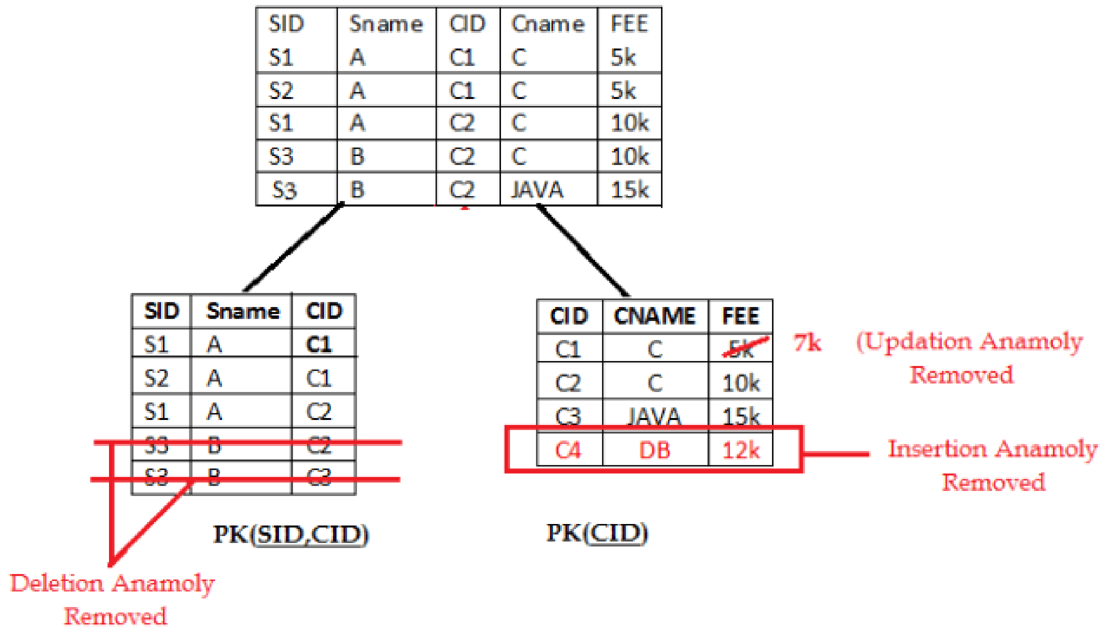
Deletion of S3 student cause the deletion of course.

Because of deletion of some data forced to delete some other useful data.



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

Solutions To Anomalies : Decomposition of Tables – Schema Refinement



SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

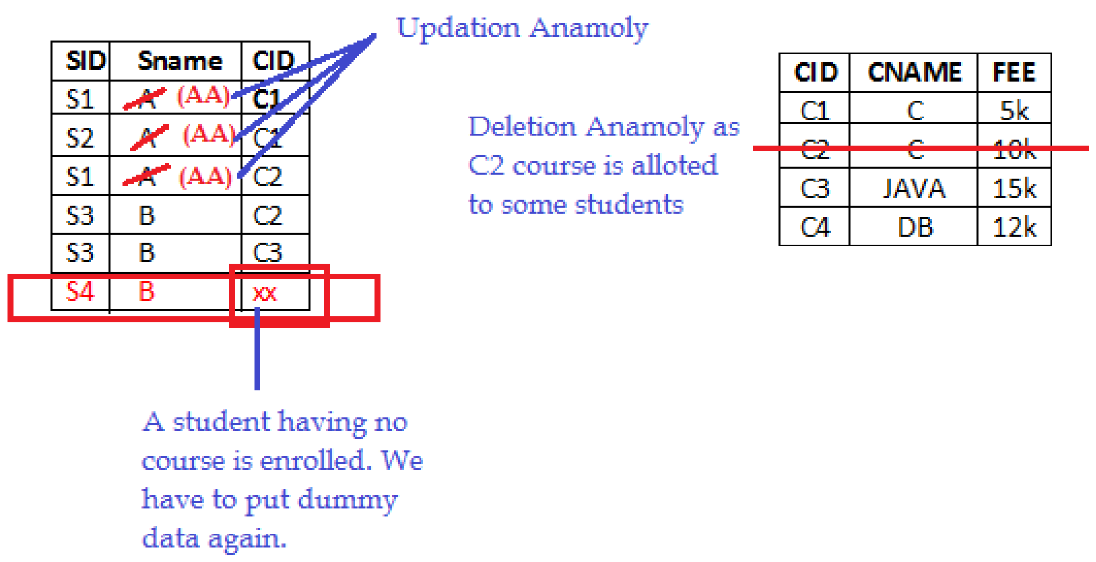
SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
~~S3~~	~~B~~	~~C2~~
~~S3~~	~~B~~	~~C3~~

PK(SID,CID)

CID	CNAME	FEE
C1	C	~~5k~~ 7k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

There are some Anomalies in this again –



SID	Sname	CID
S1	~~A~~ (AA)	C1
S2	~~A~~ (AA)	C1
S1	~~A~~ (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

Updation Anamoly

A student having no course is enrolled. We have to put dummy data again.

Deletion Anamoly as C2 course is alloted to some students

CID	CNAME	FEE
C1	C	5k
~~C2~~	~~C~~	~~10k~~
C3	JAVA	15k
C4	DB	12k

What is the Solution ??

Solution : decomposing into relations as shown below

R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

→**TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement technique called **DECOMPOSITION.**

Decomposition:- Process of decomposing a larger relation into smaller relations.

→Each of smaller relations contain subset of attributes of original relation.

Functional dependencies:

→Functional dependency is a relationship that exist when one attribute uniquely determines another attribute.

→Functional dependency is a form of integrity constraint that can identify schema with redundant storage problems and to suggest refinement.

→A functional dependency A→B in a relation holds true if two tuples having the same value of attribute A also have the same value of attribute B

IF t1.X=t2.X then t1.Y=t2.Y where t1,t2 are tuples and X,Y are attributes.

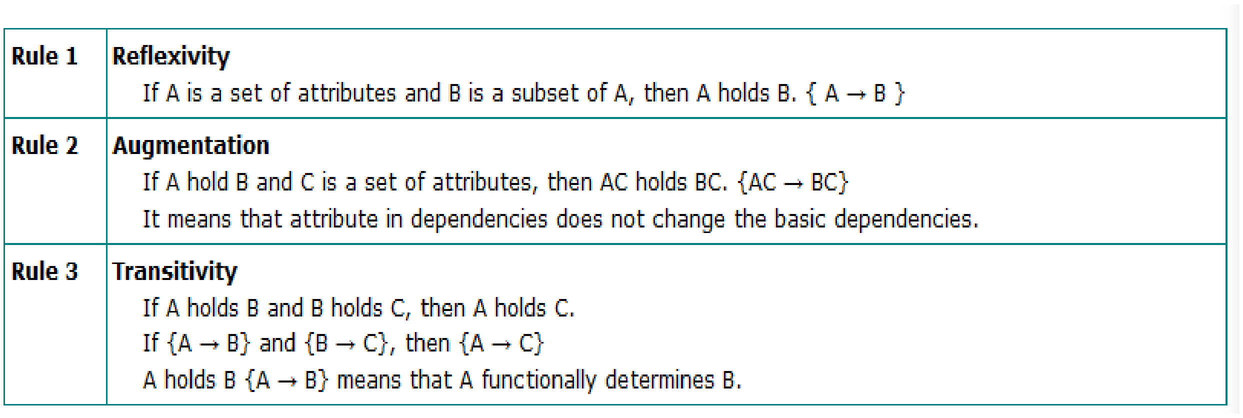
Reasoning about functional dependencies:

Armstrong Axioms :

Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

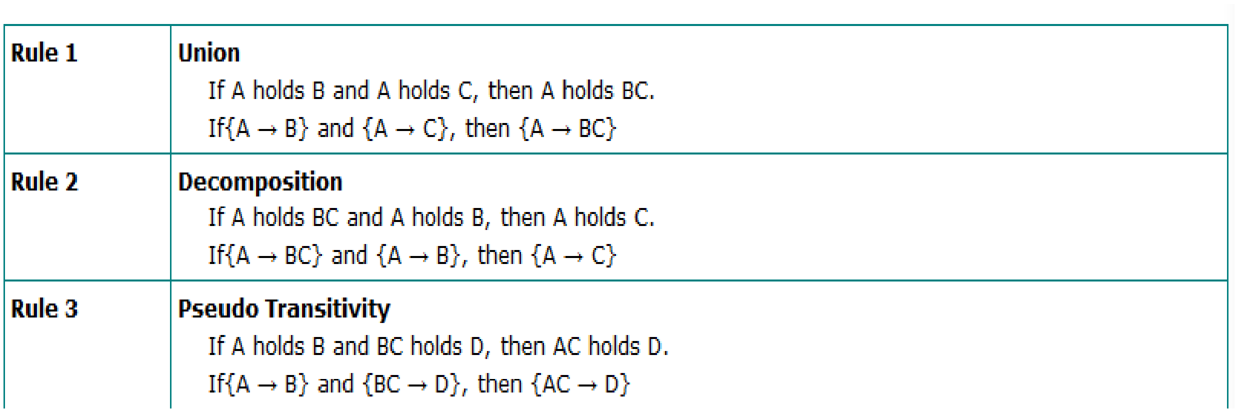
Various axioms rules or inference rules:

Primary axioms:



Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

secondary or derived axioms:



Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Attribute closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

NOTE:

To find attribute closure of an attribute set-

1)add elements of attribute set to the result set.

2)recursively add elements to the result set which can be functionally determined from the elements of result set.

Algorithm : Determining X^+, the closure of X under F.

```
Input : Let F be a set of FDs for relation R.

Steps:
            1. X⁺ = X                              //initialize X⁺ to X
            2. For each FD : Y -> Z in F Do
               If Y ⊆ X⁺  Then                     //If Y is contained in X⁺
               X⁺  =   X⁺  ∪  Z                    //add Z to X⁺
               End If
               End For
            3. Return  X⁺                        //Return closure of X

Output : Closure  X⁺ of X under F
```

Types of functional dependencies:

1)**Trivial functional dependency:-**If X→Y is a functional dependency where Y subset X, these type of FD's called as trivial functional dependency.

2)Non-trivial functional dependency:-If X→Y and Y is not subset of X then it is called non-trivial functional dependency.

3)Completely non-trivial functional dependency:-If X→Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency.

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

Candidate Key:
Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
Consider student table: student(<u>sno</u>, sname,sphone,age)
we can take **sno** as candidate key. we can have more than 1 candidate key in a table.
types of candidate keys:
1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)
Super Key:
Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(<u>sno</u>, sname,sphone,age)
we can take sno, (sno, sname) as super key

Finding candidate keys problems:

Example 1: Find candidate keys for the relation R(ABCD) having following FD's

$AB \rightarrow CD$, $C \rightarrow A$, $D \rightarrow A$

Solution:

$AB^+ = \{ABCD\}$ A and B are prime attributes

$C \rightarrow A$ replace A by c

$BC^+ = \{ABCD\}$ A and C are prime attributes ($A^+ = A^+ = \{AC\}$)

$D \rightarrow B$ replace B by D

$AD^+ = \{ABCD\}$ A and D are prime attributes ($D^+ = \{BD\}$)

$CD^+ = \{ABCD\}$ (replacing A by C in AD)

AB, BC, CD, AD are candidate keys.

Example 2: Find candidate keys for R(ABCDE) having following FD's

$A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A$

Solution:

$A^+ = \{ABCDE\}$ A is candidate key and prime attribute

$E \rightarrow A$ so replace A by E

$E^+ = \{ABCDE\}$ E is candidate key and prime attribute

$CD \rightarrow E$ replace E by CD

$CD^+ = \{ABCDE\}$ ($C^+ = C$ and $D^+ = D$) no proper subset of CD is superkey. so CD is candidate key

$B \rightarrow D$

$BC^+ = \{ABCDE\}$ ($B^+ = BD$) BC is candidate key

A, E, CD, BC are candidate keys

Question 1 : Given a relation R(ABCDEF) having FDs {AB→C, C→D, D→E , F→B, E→F} Identify the prime attributes and non prime attributes .

Solution :

$(AB)^+$: {ABCDEF} ⇒ Super Key

$(A)^+$: {A} ⇒ Not Super Key

$(B)^+$: {B} ⇒ Not Super Key

Prime Attributes : {A,B}

(AB) → Candidate Key

↓ (as F → B)

$(AF)^+$: {AFBCDE}

$(A)^+$: {A} ⇒ Not Super key

$(F)^+$: {FB} ⇒ Not Super Key

(AF) → Candidate Key

↓

$(AE)^+$: {AEFBCD}

$(A)^+$: {A} ⇒ Not Super key

$(E)^+$: {EFB} ⇒ Not Super key

(AE) → Candidate Key

↓

$(AD)^+$: {ADEFBC}

$(A)^+$: {A} ⇒ Not Super key

$(D)^+$: {DEFB} ⇒ Not Super key

(AD) → Candidate Key

↓

$(AC)^+$: {ACDEFB}

$(A)^+$: {A} ⇒ Not Super Key

$(C)^+$: {DCEFB} ⇒ Not Super Key

⇒ Candidate Keys {AB, AF, AE, AD, AC}

⇒ Prime Attributes {A,B,C,D,E,F}

⇒ Non Prime Attributes {}

Question 2: Given a relation R(ABCDEF) having FDs {AB → C, C → DE , E → F, C → B} Identify the prime attributes and non prime attributes.

Solution :

$(AB)^+$: {A B C D E F}

$(A)^+$: {A}

$(B)^+$: {B}

(AB) ⇒ (AC), $(AC)^+$: {ABCDEF}

$(C)^+$: {DECBF}

⇒ Candidate Keys {AB, AC}

⇒ Prime Attributes {A,B,C}

⇒ Non Prime Attributes {D,E,F}

Normalization:

Normalization is a process of designing a consistent database with minimum redundancy which support data integrity by grating or decomposing given relation into smaller relations preserving constraints on the relation.

→Normalisation removes data redundancy and it will helps in designing a good data base which involves a set of normal forms as follows -

1)First normal form(1NF)

2)Second normal form(2NF)

3)Third normal form(3NF)

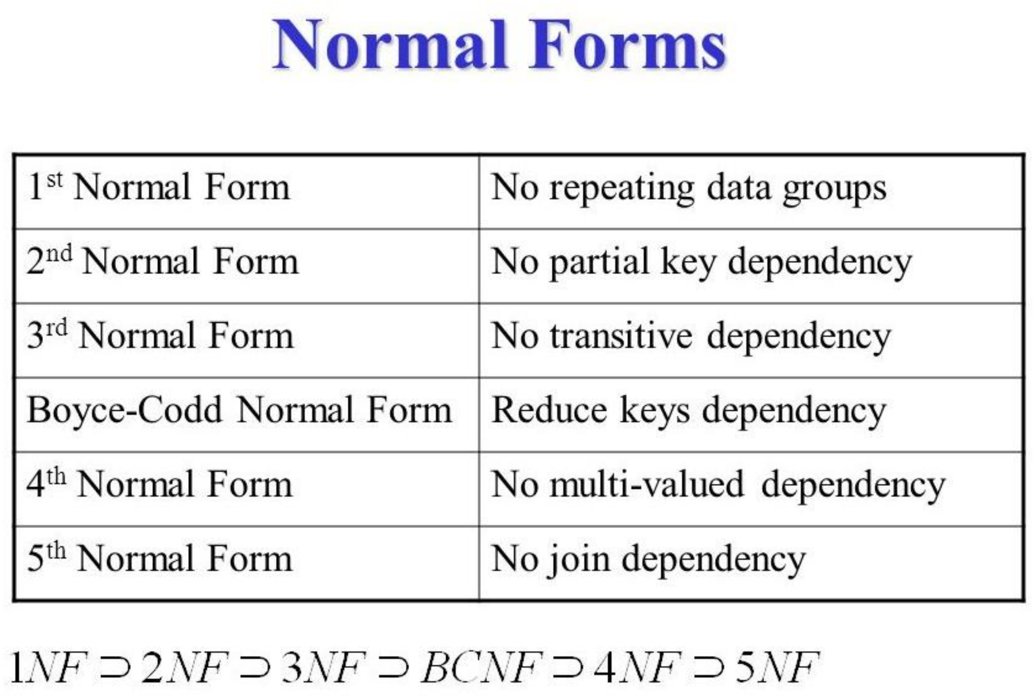
4)Boyce coded normal form(BCNF)

5)Forth normal form(4NF)

6)Fifth normal form(5NF)

7)Sixth normal form(6NF)

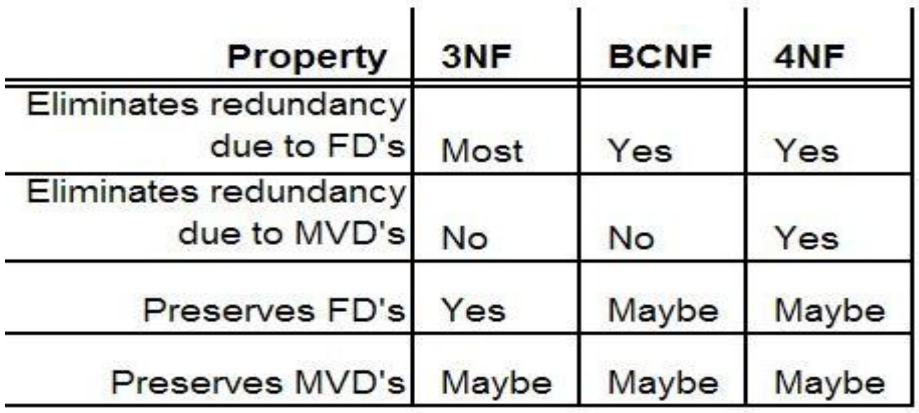
8)Domain key normal form.



Normal Forms

1st Normal Form	No repeating data groups
2nd Normal Form	No partial key dependency
3rd Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4th Normal Form	No multi-valued dependency
5th Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$



Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	Most	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	Maybe	Maybe
Preserves MVD's	Maybe	Maybe	Maybe

Properties of normal forms and their decompositions

1)First normal form: A relation is said to be in first normal form if it contains all atomic values or single values.

Example:

Domain	**Courses**
Programming	C , java
Web designing	HTML , PHP

The above table consist of multiple values in single columns which can be reduced into atomic values by using first normal form as follows-

Domain	**Courses**
Programming	C
Programming	Java
Web designing	HTML
Web designing	PHP

2)Second normal form: A relation is said to be in second normal form if it is in first normal form without any partial dependencies.

→In second normal form non-prime attributes should not depend on proper subset of key attributes.

Example:

Student id	**Student name**	**Project Id**	**Project name**

Here (student id, project id) are key attributes and (student name, project name) are non-prime attributes. It is decomposed as-

Student id	Student name	Project id

Project id	Project name

3)Third normal form: A relation is said to be in third normal form , if it is already in second normal form and no transitive dependencies exists.

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Student id	Student name	City	country	ZIP

It is decomposed as:

Student id	Student name	ZIP

ZIP	city	country

4)Boyce normal form: It is an extension of third normal form where in a functional dependency X→ A , X must be a super key.

A relation is in BCNF if in every non-trivial functional dependency X –> Y, X is a super key.

5)fourth normal form: A relation is said to be in fourth normal form if it is in third normal form and no multi value dependencies should exist between attributes.

Note: In some cases multi value dependencies may exist not more than one time in a given relation.

6)fifth normal form: fifth normal form is related to join dependencies.

→A relation R is said to be in fifth normal form if for every join dependency JD join {R_1 , R_2 ,…….R_N } that holds over relation R one of the following statements must be true-

1)R_i =R for some i

2)the join dependency is implied by the set of those functional dependency over relation R in which the left side is key attribute for R.

NOTE: if the relation schema is a third normal form and each of its keys consist of single attribute, we can say that it can also be in fifth normal form.

→A join dependency JD join {R_1, R_2, ……R_N} is said to hold for a relation R if R_1,R_2…..R_N this decomposition is a loss less join decomposition of R.

→When a relation is in forth normal form and decompose further to eliminate redundancy and anomalies due to insert or update or delete operation, there should not be any loss of data or should not create a new record when the decompose tables are rejoin.

7)Domain key normal form: A domain key normal form keeps a constraint that every constraint on the relation is a logical sequence of definition of keys and domains.

8)Sixth normal form: A relation is said to be in sixth normal form such that the relation R should not contain any non-trivial join dependencies.

→ Also sixth normal form considers temporal dimensions(time) to the relational model.

Key Points related to normal forms –

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

problems on normal forms:

Problem 1:

Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC --> D
CD --> AE

Solution:

Important Points for solving above type of question.
1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.
Candidate keys in given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2 NF. S**o, the highest normal form is 1 NF.**

problem 2:

Find the highest normal form of a relation R(A,B,C,D,E) with

FD set as

{BC->D,
AC->BE,
B->E}

Step 1: As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
Step 2: Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.
Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.
The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.
So the highest normal form of relation will be 2nd Normal form.

Decomposition: It is the process of splitting original table into smaller relations such that attribute sets of two relations will be the subset of attribute set of original table.

Rules of decomposition:

If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1)Union of two smaller subsets of attributes gives all attributes of 'R'.

R1(attributes)UR2(attributes)=R(attributes)

2)Both relations interaction should not give null value.

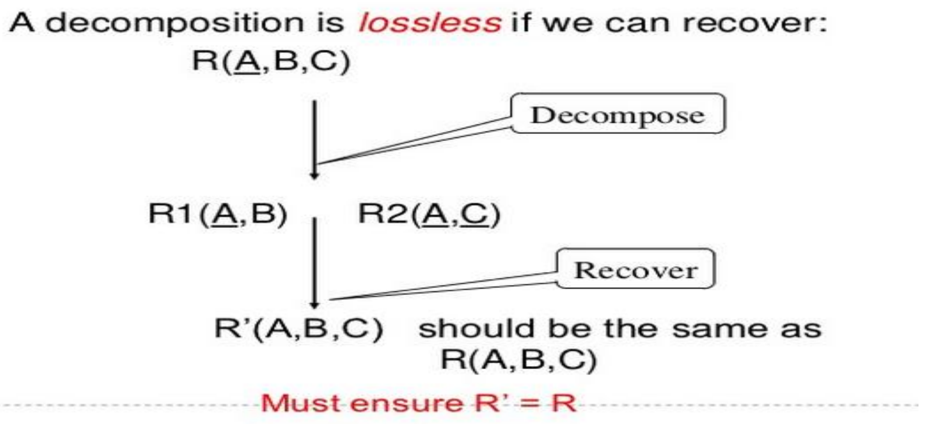
R1(attributes)∩R2(attributes)!=null

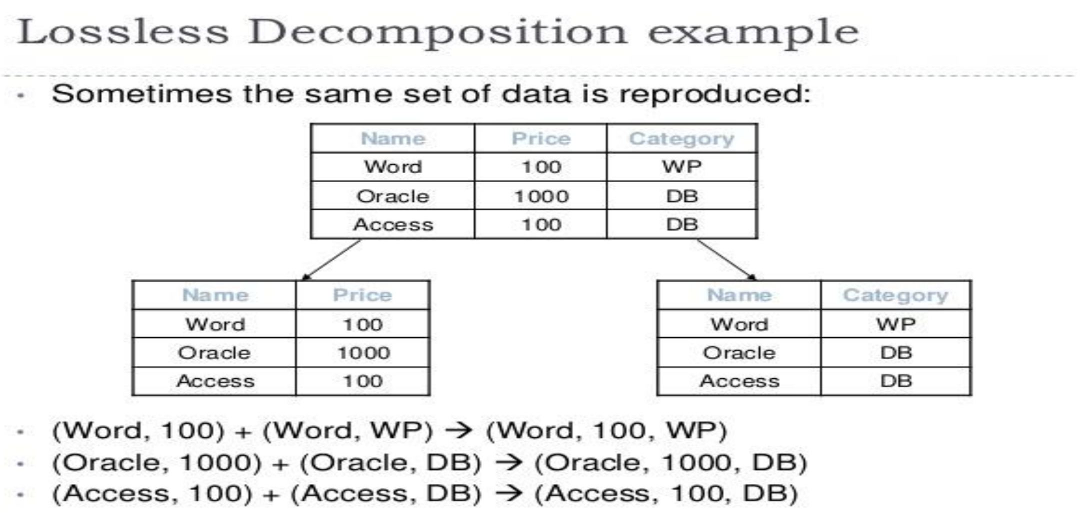
3) Both relations interaction should give key attribute.

R1(attribute)∩R2(attribute)=R(key attribute)

Properties of decomposition:

Lossless decomposition: while joining two smaller tables no data should be lost and should satisfy all the rules of decomposition. No additional data should be generated on natural join of decomposed tables.


A decomposition is *lossless* if we can recover:
R(A,B,C)
Decompose
R1(A,B)
R2(A,C)
Recover
R'(A,B,C) should be the same as R(A,B,C)
Must ensure R' = R



Lossless Decomposition example

- Sometimes the same set of data is reproduced:

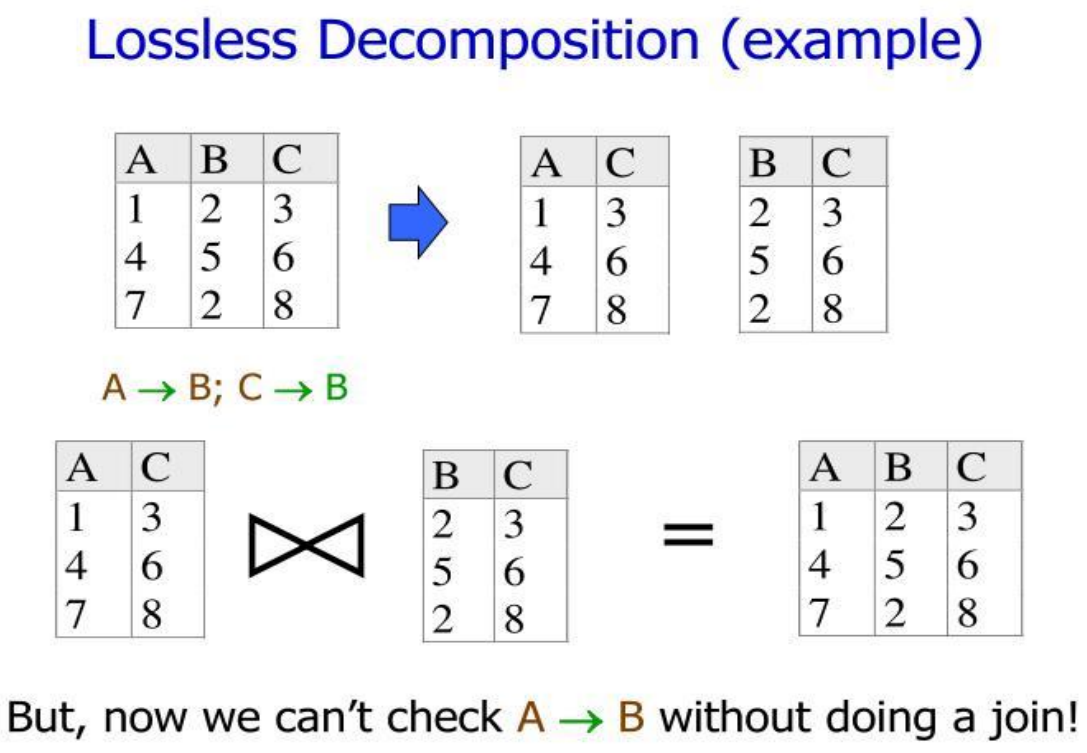
Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100

Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)

example 2 for loseless decomposition:



Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B; C \rightarrow B$

A	C
1	3
4	6
7	8

$\bowtie$

B	C
2	3
5	6
2	8

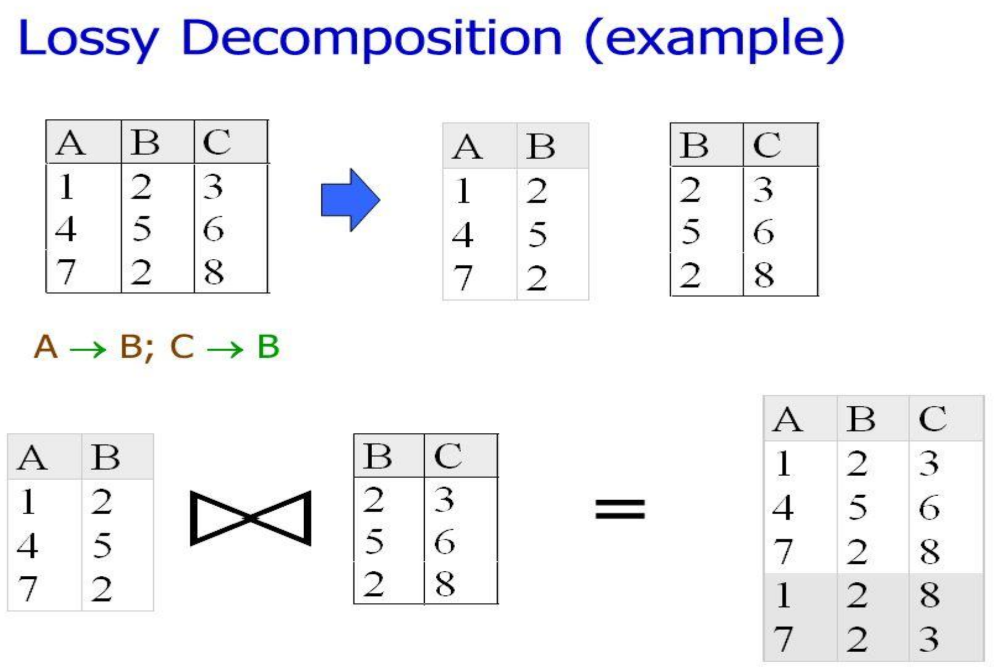
$=$

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Lossy join decomposition: if information is lost after joining and if do not satisfy any one of the above rules of decomposition.

example 1:



Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B; C \rightarrow B$

A	B
1	2
4	5
7	2

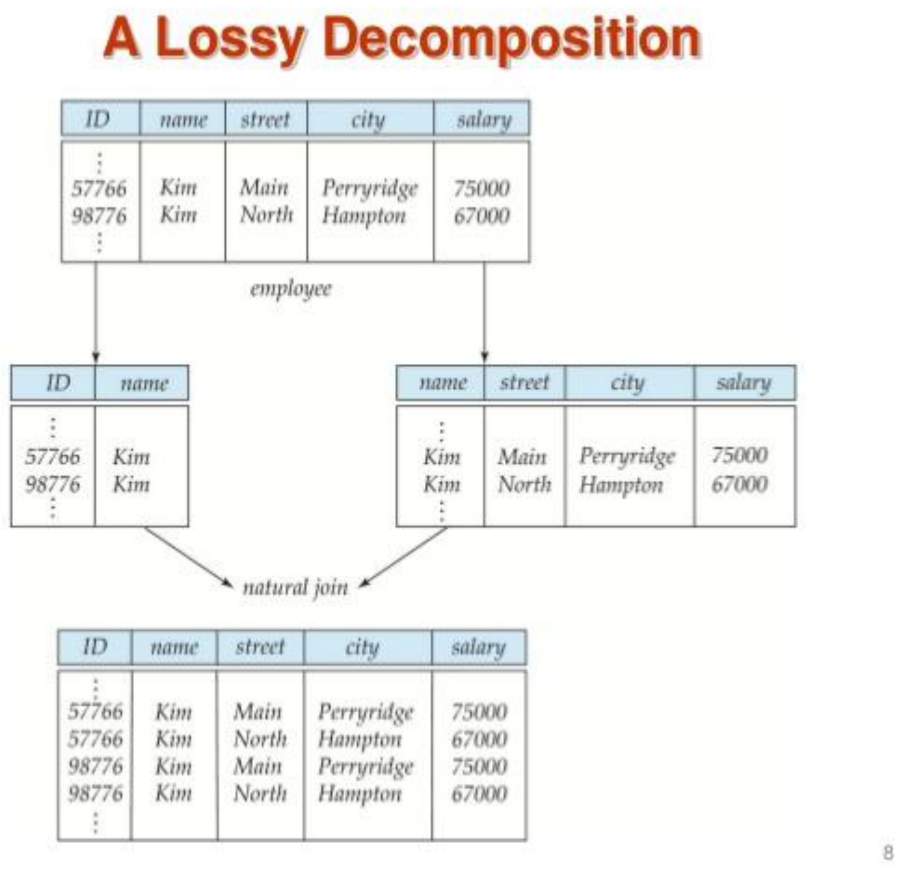
$\bowtie$

B	C
2	3
5	6
2	8

$=$

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

example 2:



A Lossy Decomposition

employee

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

8

In above examples, on joining decomposed tables, extra tuples are generated.

so it is lossy join decomposition.

Dependency preservation: functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables.

Dependency Preservation

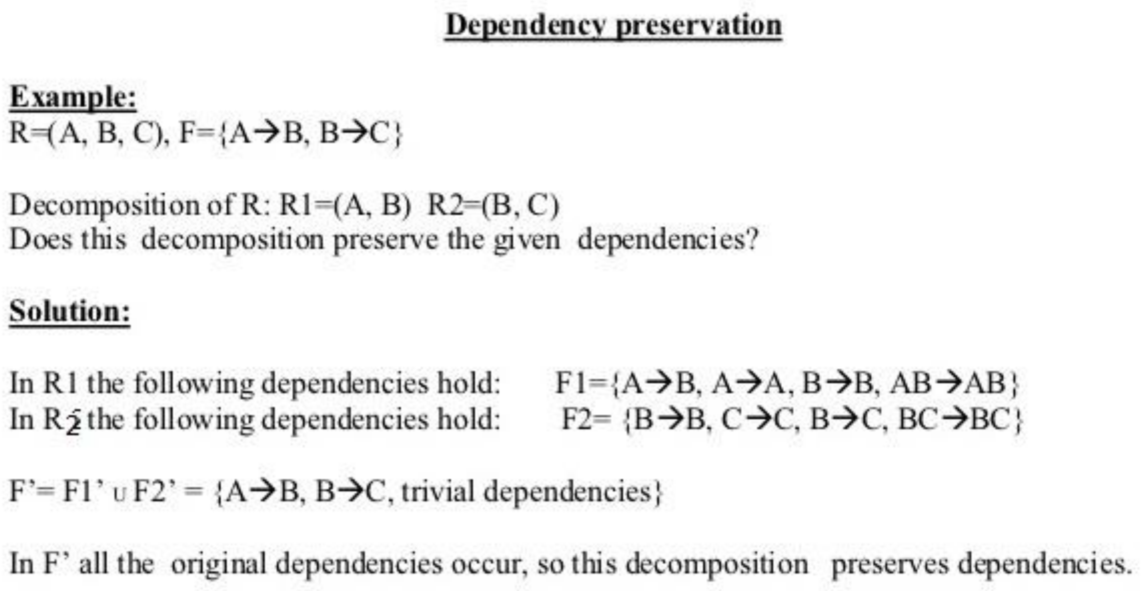
A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

```
(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then
there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.
```

example for dependency preservation:



Dependency preservation

Example:
R=(A, B, C), F={A→B, B→C}

Decomposition of R: R1=(A, B) R2=(B, C)
Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold: F1={A→B, A→A, B→B, AB→AB}
In R2 the following dependencies hold: F2= {B→B, C→C, B→C, BC→BC}

F'= F1' ∪ F2' = {A→B, B→C, trivial dependencies}

In F' all the original dependencies occur, so this decomposition preserves dependencies.

lack of redundancy: It is also known as repetition of information. The proper decomposition should not suffer from any data redundancy.