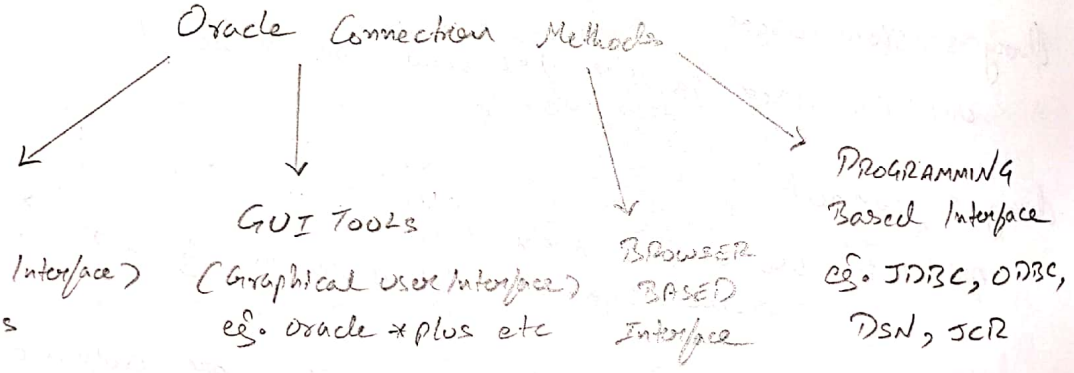
ORACLE DATABASE MANAGEMENT SYSTEM [LAB WORK] :-

ORACLE is one of the most popular & widely used RDBMS by most of the top most business enterprises. It comes in different variation and versions such as oracle 9i, 10g, 11g, 12c etc in all these versions SQL part is more or less same but other features are gradually added. oracle DBMS comes in two variations :-

(i) Enterprise Edition (ii) Express Edition
<-> (Paid) <-> (foss i.e free or open source software)

=> ORACLE INTERFACE METHODS :- These are different ways to connect to an oracle server. Methods are shown below :-



(i) Command Line Interface :- It is the fastest type of interface available for Database server connection and it can be done either ~~directly~~ directly on the server or through some network based interface like Putty or VPN.

(ii) GUI Tools :- These are different types of GUI Tools available to connect to oracle Database Server one inbuilt tool is SQL *plus.

(iii) Browser Based Interface :- Starting from version 9i oracle has started the support of browser based connectivity to oracle Database server. Browser can usually communicate with a web server such as IIS (Internet information Services), Tomcat etc it means that the must be an

By only creating user will not allow you to login until you have permission to login. This permission can be given by parent user only i.e system or sys.

⇒ To Grant login permission, steps to follow are :-

(i) login as system or sys
(ii) check with show user
(iii) Grant resource, connect to MCA2019SECB68 ;
username

Note :- The new user which will be created will have no tables or records in it. it will be empty.

⇒ Data storage in DBMS / RDBMS :-

As we know that in DBMS / RDBMS, user data or application data is store inside table which is a two dimensional structure consisting of rows and columns. however, different companies used different methods to store tables and other database objects.

For example :- In case of MS Access, an OS level single file is created with an extension of .mbd or .accdb, this file contains tables, forms, queries etc in a single file in an encapsulated and abstracted method.

We can view tables, forms, queries etc at a single place after opening this .mbd file. However, there are lot of security issues with this method. However, in case of more robust RDBMS like oracle or DB2 data is kept in much more secure manner.

oracle keeps user data under a user name & you can view your data only if you have permission & password for that user. however, since data is to be stored permanently it must be stored in a file but you will never find a file name matching a table name.

intermediate webserver between browser & Database Server.

(iv) Programming based interface :- This method of Database connectivity is highly widely used by programmers & Developers to connect there application interface with the database. It requires use of drivers & connection string etc.

Points to Remember :- Localhost is an alias for local loop back address i.e 127.0.0.1 . 10.0.0.1 or 127.0.0.1 or 172.16.0.1 all there ip addresses are reserved for local loop back. URN stands for (Uniform Resource Name)
5560 is oracle default port no.

⇒ ORACLE COMMAND's :-

Query : show user
OUTPUT :- user is "" [it show logged in user]

Query : Connect
OUTPUT :- username : ***
password : *** [it is used to log in]
⇒ system / sys
⇒ orcl or oracle

Note :- when you install oracle some users are automatically created such as sys, system, HR. the first two users belong to DBA category.

Query : Select * from tab / table ; [it is used to see the list of table]
The output screen displays around 200 or low + tables, all these tables are data dictionary table reserved for oracle internal working therefore we will create our new user.

Note :- To create new user you must be login as system or sys.

Query : Create user MCA2019SECB68 identified by vijay ;
username password

the oracle adds it at the end.

(06) No two columns can ever have same name.

(05) A table can have only one primary key.

(07) A table can have many foreign keys.

(08) Any columns or any row can be removed without effecting other data stored in the Database.

point to remember :- Always write name in quotes in upper case.

=> PSEUDO TABLES And PSEUDO COLUMNS :-

Oracle provides few pseudo columns and tables for different use.

for example :- "DUAL" table is a dummy table which is used for different purposes. The Dual table can be used for the following purposes i.e,
- → To view current user
- → To view current date
- → To perform any mathematical operation

queries >> Select user from dual; // to view current user
- >> Select sysdate from dual; // to view current date
- >> Select 6453 x 345 from dual; // to perform Mathematical operations

Dual table is a pseudo table or data dictionary table which is not displayed in the user table list.

To view list of tables following queries can be used :-

```
>> select * from tab;    // it shows tables like 8 row
>> select * from cat;    // it shows some extra table like 11 row
>> select * from user_tables;  // it show all tables in expanded form
```

NOTE : COLUMN NAME of DUAL TABLE IS DUMMY.

When we install oracle all it's files are extracted in computer drive C. when you reach to the folder C:/oracle/product/10.2.0 you can see three types of files .CTL, .DBF and .LOG. The .DBF files contains all user data including tables however these table can never be viewed at OS level, the .CTL files are control files which are used for setting oracle server parameters and .LOG files contains all transaction logs.

In RDBMS table stores user data as well as oracles internal data which are known as Meta data. when you connect with either sys or system user the table that are displayed are actually Meta data tables or Data Dictionary tables. These Tables can be viewed but in most cases a normal user can not insert or modify these data. For example :- If you want to see the list of all users

query >> Select * from all_users;

OUTPUT : LIST of TABLES STORED IN THE DATABASE.

All these tables as well as user tables are based on codd rules with having following properties :-

(01) A table columns are always displayed in upper case

(02) The column of a table can be viewed in any order. however, a new column is always added at the end.

For example :- SELECT * FROM EMP;

OUTPUT :

ECODE	ENAME	SALARY	DCODE
=	=	=	=

(03) The default order of a column is the creation time order

(04) The rows of a table can be viewed in any order however the default order is insertion order. whenever a new row is created

Note :- Describe is an internal command that is why it can be abbreviated.

=> INSERT Statement :-

The Insert statement is used to insert data / records in an table. It can never be used to update any existing value or to fill an empty column. it always adds a new record after previous added record. Insert statement is a DML statement (Data Manipulation language).

List of DML statements :-

-> Insert -> Delete
-> update -> Select

There are different ways to insert data in a table :-

(i) Inserting Data in all columns :-

>> Insert into sample values (11, 'AMIT');
table name values to be insert

OUTPUT : 01 row created

(ii) Inserting values in few columns :- → written in uppercase in quotes

>> Insert into Sample (Name) values ('MUDIT');
Column name value to be inserted

(iii) Inserting values in few columns using NULL keywords :-

>> Insert into Sample values (NULL, 'ABCD');
>> Insert into sample values (666, NULL);

=> DCL Statement (Data Control language) :-

DCL statements are used to controll the behaviour of transaction like commit or roll back.

For Example :- If you accidently closed your working window / login window then all your data might get lost.

these three tables are not displayed in user table list because these are Data dictionary tables or DD tables. Data Dictionary tables are not displayed in user login but user can access them and they are part of sys & System Schema.

=> DDL Statements (Data Defination Language) :-

Data Defination language queries are those queries which are related to the structure or schema of the table or user. List of statements that come under DDL are :-

(i) CREATE (ii) ALTER (iii) TRUNCATE (iv) DROP

Note :- In all DDL command table is written before table name.

=> CREATE Statement :-

Create is a versatile DDL statement which can be used to create following -> user -> Alias / Synonyms
-> Table -> Sequences
-> view -> Index

table can be created in different ways :-

=> Creating table at column level :-

Query >> Create Table Sample (Rollno Number, Name varchar2(10));

To view the structure of the table which is also known as Schema or design definition or structure Describe command is used :

Query >> Describe tablename ; or >> DESC tablename

OUTPUT :

Name	NULL ?	TYPE
RollNO		Number
Name		varchar2 (10)

→ Changing the structure of a table :-

The structure of the table can be changed using alter table DDL command and it includes following operations :-

(i) adding a column.
(ii) removing a column.
(iii) changing the size of a column.
(iv) changing the datatype of a column.

⇒ Adding a column :-

Query >> Alter table sample add (email varchar2 (10));

⇒ Removing a column :-

Query >> Alter table sample drop column phoneno;

⇒ Changing the size of a column :- The size of column can be increased or decreased at any time however while decreasing error can come if you try to decrease the size of a column smaller than any current largest values.

Query >> Alter table sample modify (name varchar2 (10));

⇒ Changing the datatype of a column :- The datatype of a column can be change only if the column is empty.

Query >> Alter table sample Modify (rollno varchar2 (10));

→ Removing Records of a table :-

The records of a table can be removed by using both DDL & DML commands. for example :- The DML statement delete can remove records collectively or selectively.

⇒ The commit statement permanently stores your data upto the current timeline which cannot be further rolled back.

⇒ The Rollback statement undo all your transactions upto the last commit point or upto the last DDL query.

Note :- All the DDL commands are auto commit that is why they can't be rolled back.

→ Inserting values in a table using Substituting method :-

Query >> Insert into Sample value (&rn, '&naam');

OUTPUT :- Enter value for rn : ....
Enter value for naam : ....

Note :- '/' back slash, it can be used to repeat last SQL query.

→ Updating values in a table :-

The values in a table can be updated or set to NULL by using update statement. More than one column can be updated using a single statement. For Example :-

⇒ To change all the values of the table :-

Query >> update Sample set phoneno = 7777;
>> update Sample set name = NULL;
>> update Sample set name = 'ABCD' where rollno = 1;

⇒ To update multiple columns :-

Query >> update Sample set phoneno = 666, rollno = 888 where rollno = 99;

➤ Environment Variables :-

Oracle provides number of environment variables to control user output we can change any of these variable values but the changes remain effective only till current login window is not closed.

To see the list of all environment variable :

```
Query >>
  |--->> set linesize 100   // used to set linesize
  |--->> set pagesize 100   // used to set pagesize
  |--->> set pause on       // used to pause at certain no. of rows
```

→ Creating table using another table :-

A table can be created from another table to either rectify some errors or for backup or anyother purpose.

(i) Creating an Exact copy :- Exact copy means that new table will have same number of rows & columns but constraints will not be copied.

```
Query >> Create table copy as select * from course ;
                  |-> new table          |-> old table
```

(ii) Creating an Exact copy with selected records :-

```
Query >> Create table copy as select * from course where coursesem>6;
```

Note :- Copy of a table is created in 2 steps i.e first of all columns are created then records are copied as per condition. However, this is a DDL operation and can not be rolled back.

(iii) Creating a table as a copy of another table without any records :-

```
Query >> create table copy as select * from course where coursesemester = -1;
  |-->> Create table copy as select * from course where 1=2 ;
```

```
Query >> Delete * from test ;
  OUTPUT : Error as you can not use * with delete command
```

Correct Query >> Delete from test ;

=> To delete particular column :-

```
Query >> Delete from test where coursecode = 101 ;
  >> Delete from test where course code = 101, 103, 104 ;
  OUTPUT : Error as multiple values cant be used to delete with delete command
    as it is an structure query language
  >> Delete from test where course code = 101 or 103 ;
  OUTPUT : Error as column name have to be specified each time.
```

Correct Query >> Delete from test where coursecode = 101 or coursecode = 103 ;

=> To delete Multiple column :-

```
Query >> Delete from test where coursecode in (105, 107, 000, 999);
```

→ TRUNCATE Statement :-

The Truncate DDL command can be used to remove all the records of a table permanently.

```
Query >> Truncate table test ;
  OUTPUT : Table truncate
```

Note :- we cannot use condition with truncate command as it is a collective not selective command.

→ Removing a table :-

```
Query >> Drop table test ;
```

Note :- Once a table is drop it cannot be rolled back.

ORACLE / SQL Constraints :- SQL provides different types of constraints to control data volume for a column. for eg :- primary key, foreign key, unique, not null, check etc.

PRIMARY KEY CONSTRAINTS :- This constraint does not allow duplicate values or null values in a column. Some properties of primary key are as follow :-

- A table can have one and only one primary key however the primary key can be made up of more than one columns.
- The primary key can be made at column level or at row level or table level.
- When primary key is made up of only one column it is known as Atomic primary key.
- When primary key is made up of more than one columns it is called composite or complex primary key.

→ Creating primary key at column level :-

>> Create table Test (Rollno Number Primary key, Name varchar2 (10));

→ Creating primary key at table level or row level :-

>> Create table Test (Rollno Number, Name varchar2 (10), phno Number, Primary key (Name));

→ Creating composite primary key :-

The above two examples were of atomic primary key but some times we may need to create primary key made up of multiple columns.

For example :- Your train ticket has only one PNR Number but there are multiple passengers.

(iv) creating a copy of a table with different column order :-

query >> create table copy as select course code, course fees, course name, course semester from course ;

(v) creating a copy of a table with different column name :-

query >> Create table copy (ccode, cname, cfees, csem) as select course code, course name, course fee, course semster from course;

(vi) creating a copy of a table with limited columns :-

query >> create table copy as select course code, course name from course;

→ Inserting data in a table from another table :-

query >> insert into my table select * from course;

Sometimes it might happen that your table might contains exact duplicate number of records which can be removed by using Pseudo column row id.

To show row id of a table :-

query >> select rowid, course code, course name from my table;

Now we can delete the duplicate rows by using these rowid

query >> Delete from my table where rowid = '1234567891...' ;

Just like rowid there is another Pseudo column row num which is assigned during run time. Rowid is permanent and it is saved with the row where as row num is tempory and is not saved within the table.

query >> select * row num, rowid from course ;

- Unique constraint can be applied on a column by using both column level or row level or table level method.

>> Create table Test (Rollno Number Primary key, Name varchar2 (10) Not NULL, Pincode Number Unique, Phno Number Not NULL, Email varchar2 (10) Unique);

Note :- Unique column can have multiple NULL values because NULL value can not be compared.

CHECK CONSTRAINTS :-

The check constraint is used to specify the range of values in a particular column of a table.

>> Create table Sample (Rollno Number, Name varchar2 (10) Unique, Pincode Number Not NULL, AGE Number check (Age > 20));

>> Describe Sample ;

OUTPUT :

Name	NULL ?	TYPE
Rollno		Number
Name		varchar2 (10)
Pincode	Not NULL	Number
Age		Number

>> Create Table Sample (Rollno Number, Name varchar2 (10) Unique, Pincode Number Not NULL, AGE Number Check (Age > 20 and Age < 70);

The above is a example to allow a range of value only in a column.

similarly, at a fast food restaurant you may have a single order number containing multiple items.

>> Create Table SAMPLE (Orderno Number Primary key, Itemcode Number Primary key, Qty Number);

Error : Table can have only one primary key !

So to create Composite Primary key only table level or row level method can be used :-

>> Create Table SAMPLE (orderno Number, Itemcode Number, Qty number, Primary key (orderno, itemcode));

Not NULL CONSTRAINTS :-

- A Not Null constraint does not allow NULL values in a column but it can allow duplicate values.
- There can be many Not NULL columns in a table.
- The Not NULL constraints can be applied only at column level.

>> Create table Test (Rollno Number Primary key, Name varchar2 (10) Not NULL);

Note :- We cannot tell whether a column is a primary key or not just by looking at the table structure.

UNIQUE CONSTRAINTS :-

- A unique constraint does not allow duplicate values in a column but it can allow null values.
- There can be many unique columns in a table
- The unique constraint is not displayed in a structure of a table.

→ the full name is Constraint_Type it only showing first character

OUTPUT :

CONSTRAINT_NAME	C	TABLE_NAME	SEARCH_CONDITION
SYS_C007481	C	SAMPLE	Pan_no > 100
SYS_C007482	P	SAMPLE	
SYS_C007483	U	SAMPLE	

The Second column constraint_type displays few constraints characters.

For Example : C → NOT NULL OR CHECK U → UNIQUE
P → PRIMARY KEY R → FOREIGN KEY

→ Removing Constraints from the Column :-

There are different ways to remove constraints from the columns

> ALTER TABLE MYTABLE DROP PRIMARY KEY ;

> ALTER TABLE MYTABLE DROP CONSTRAINT SYS_C007484 ;
(SYS_C007484 ↓ constraint Name)

→ ASSIGNING NAMES TO CONSTRAINTS :-

Oracle provides internal Constraint names beginning with SYS_ however, we can also specify our name.

>> CREATE TABLE MYTABLE (RollNo NUMBER CONSTRAINT RNO_PK PRIMARY KEY, NAME VARCHAR2(10) CONSTRAINT UNI UNIQUE , EMAIL NUMBER CONSTRAINT NN NOT NULL) ;

Now, we can remove constraint using the Assigned name

>> ALTER TABLE MYTABLE DROP NN ;

→ we can apply multiple constraints on a s

>> Create Table Sample (RollNo Number Primary key, Name varchar2(10) Not NULL unique , Pincode Number Unique);

As we can see that some of the constraints are not visible in the table structure these constraints can be seen at two places that are
(i) software SRS document (ii) Data Dictionary table

Data Dictionary :- As we know that oracle maintains a very comprehensive data dictionary consisting of thousands of tables.

Example :- > Select * from All_users ;

Similarly, there is another table known as User_Constraints which stores details of all constraints applied on all columns of tables.

> Select * from USER_CONSTRAINTS ;

The above query displays no. of columns that are not required therefore we need to narrow down our search results to do this first of all discribe user_constraints.

> Describe user_constraints ;

We will select only initial 05 columns from user_constraints.

> Select Constraint_name, Constraint_Type, Table_name, Search_condition from user_Constraints where table_name = 'SAMPLE' ;

Note :- Table name will always be in capital letter as the table name stored in Data Dictionary is in capital or uppercase.

As we can see that there are two entities therfore at least two tables will be created.

first, we will create customer table :-

```
>> Create table customer (Ccode Number, CName varchar2 (10),
   CAddress varchar2 (10));
>> Insert into customer values (1, 'kapil', 'Delhi');
>> Select * from customer;
```

OUTPUT:

Ccode	CName	CAddress
1	kapil	Delhi
2	Mohit	Dehradun
3	Mudit	Mumbai

Second, we will create Account table :-

```
>> Create table Account ( AcNumber Number, Actype varchar2(10),
   Balance Number );
>> Insert into Account values ( 101, 'saving', '7777');
>> Select * from Account;
```

OUTPUT:

Ac Number	ACTYPE	BALANCE
101	Saving	7777
201	saving	5674
301	current	5544

So, for two entity boxes there are two tables however the above table design is not correct; because by looking at the account table you cannot tell that which account belongs to which customer. it is because we have ignored the relationship box.

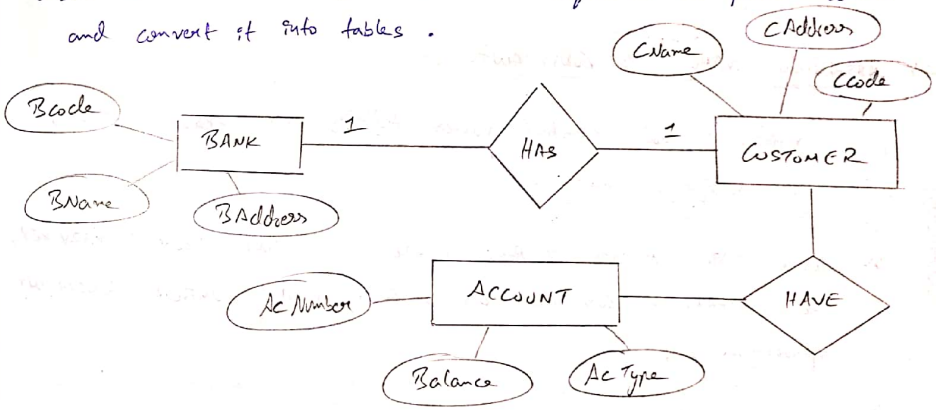
Foreign key :- The foreign key constraint is also known as referencial integrity constraint. An attribute is termed as foreign key if it is dependend on another attribute for its value.

In other words, if a column is foreign key then it can contain only those values which are present in its parent dependent column. Therefore, sometimes a foreign key column is also known as child column.

For example :- A bank has number of customers and these customer can avail different type of banking services like,

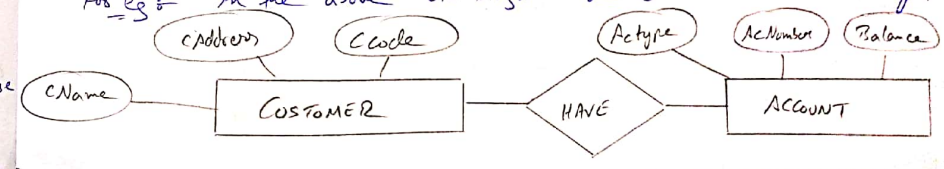
- they can open bank account
- they can rent a locker
- they can open an FD or RD account

Ques: Draw an ER Diagram for above requirement or problem context and convert it into tables.



As a general rule the entity for which an ER diagram is being created is not shown in the diagram.

For eg :- In the above ER Diagram we can avoid bank entity.



>> Select * from customer ;

OUTPUT :

	Ccode	Cname	Caddress	
Entity Sets	1	Kapil	Delhi	→ Entity Instance 1
	2	Mohit	Dehradun	→ Entity Instance 2
	3	Modit	Mumbai	→ Entity Instance 3

>> Select * from Account ;

OUTPUT :

	Ac Number	Ac Type	Balance	
Entity Sets	101	Saving	7777	→ Entity Instance 1
	201	saving	5674	→ Entity Instance 2
	301	current	5544	→ Entity Instance 3

So, every row in above two table represent an entity instance or entity object or entity example.

Now, a third relationship table will be created. when a relationship table is created it borrows participating attributes from the related entities Therefore, the third relationship table will have two attributes Ccode and Ac Number i.e Primary key attributes only.

>> Create table relation (Ccode Number, Ac Number Number);

>> Select * from relation ;

OUTPUT :

Ccode	Ac Number	
1	201	one to many Relationship or cardinality
2	101	
1	301	

However, the above table design is still not correct.

We can add values of column who does not even exist in the customer and Account table this is because the column Ccode & Ac Number have not yet been made primary key.

Depending upon the type of relationship or mapping cardinality between two entities we may need to create another table to implement the relationship. for example :- If the relationship is one to many two entities can be shown with the help of two tables however if the relationship is many to many two entities we have to create three tables atleast to show relationship between them.

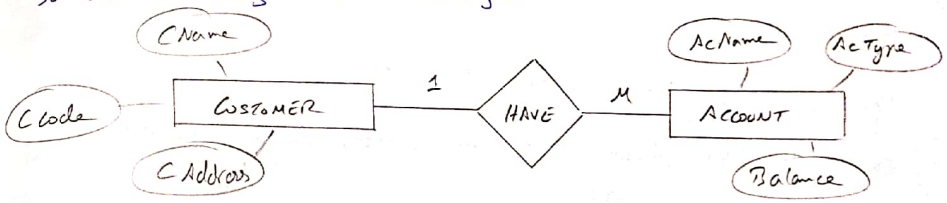
1 : 1 or 1 : M → 2 Entity → 2 Tables (Minimum)

M : M → 2 Entity → 3 Tables (Minimum)

Now let us take the case of one to many relationship.

for example :- A customer can have many accounts in the same bank

so our ER Diagram will be changed to



Since, the relationship between customer & account is one to many

So there are two options (i) Two table solution

(ii) three table solution

=> Three Table Solution in case of one to Many Relationship :-

Step (01) Create two tables for the two entities. In theory these tables will be called as entity table. for eg :- following two tables will be created

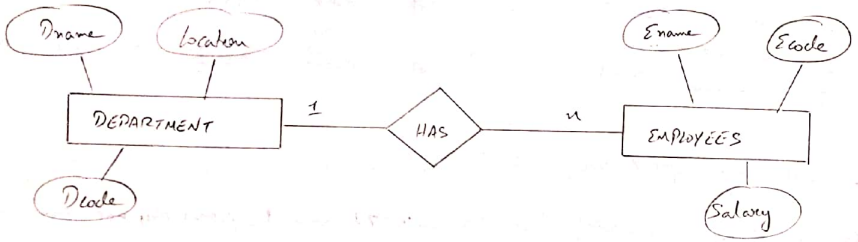
Entity table 1

> Create table customer (Ccode Number, Cname varchar(20), Caddress varchar2(10));

Entity table 2

> Create table Account (Ac Name varchar(10), Balance Number, Ac Type varchar(10));

Ques :- A company has no. of departments at different locations or city and in each department there are no. of employees. Draw an ER diagram for above problem context and convert it into tables.



Now, we will convert the above ER diagram into tables and since the cardinality between above two entities is one to many therefore there are two possible solutions :- (i) Two table Solution
(ii) Three table Solution

→ Two Table Solution (in case of one to many) :-

we will create two tables i.e Department table & Employee table,

Department table

Dcode	Dname	location
1	HR	Mumbai
2	Account	Delhi
3	Marketing	Meerut

Employe table

Ecode	Ename	Salary
101	AMIT	7878
201	KAPIL	17878

However, the above tables are incomplete because you cannot tell that which Employee belongs to which Department. It is because we have not paid attention to the relationship and since we are doing a two table solution therefore the above employee table will be redesign.

>> Create table emp4 (Code number, Ename varchar2(10), Salary number, Dcode number references Department);

Example :-

>> insert into relation values (22, 101);
>> insert into relation values (33, 999);
>> Select * from relation;

OUTPUT :

Ccode	Ac Number
1	201
2	101
1	301
22	101 → not exist in Account table
33	999

Not exist in Customer table.

Here, Customer code 33 and 22 are invalid similarly Account number 999 is also invalid.

The problem can be solved by making foreign key in relation table.

→ To create foreign key :-

>> Create table Cust_Acc (Ccode Number References Customer, Ac Number References Account);

OUTPUT : error reference table does not a primary key

>> ALTER TABLE CUSTOMER ADD PRIMARY KEY (Ccode);

>> ALTER TABLE ACCOUNT ADD PRIMARY KEY (Ac Number);

Now, if we try to create foreign key

>> Create table Cust_Acc (Ccode Number References Customer, Ac Number References Account);

OUTPUT : Table created.

To solve the duplicate values problem in three table solution we will make the column unique using constraint unique.

→ foreign key at column and row level :-

A foreign key can be created at ① Column level and
② table or row level.

⇒ At column level the foreign key keyword is not used.

» create table emp (ecode number, ename varchar2 (10), dcode number references department);

⇒ At table/row level the foreign key keyword is used.

» create table emp (ecode number, ename varchar2 (10), dcode number, salary number, foreign key (dcode) references department);

→ Apply Multiple Constraints on Primary key Column :-

Example :- The company also has no. of projects in which different employees are working and each project belong to only one department.

» create table project (dcode number primary key references department check (dcode > 2), pname varchar2 (10), cost number);

Note :- The primary key and the foreign key columns need not to be in separate tables.

» create table emp (ecode number primary key, ename varchar2 (10), mgr number references emp (ecode));

» Select * from Department; references » Select * from emp4;

	Dcode	Dname	location
parent key	1	HR	Mumbai
	2	Accounts	Delhi
	3	Marketing	Meerut

Ecode	Ename	Salary	Dcode
101	AMIT	5555	2
201	KAPIL	666	2
301	ROHIT	2222	1

Dcode — foreign key

» insert into emp4 (401, 'Mohit', 8888, 5);

Error : integrity constraint (SYS.SYS_C0075977) violated - parent key not found

→ Three table Solution (in case of one to many) :-

In this solution there will be two diff entity tables and one relationship table.

» Create table emp-depart (dcode number, ecode number);

» Select * from Department;

	Dcode	Dname	location
parent key	1	HR	Mumbai
	2	Account	Delhi
	3	Marketing	Meerut

» select * from employee;

Parent key	Ecode	Ename	Salary
	101	AMIT	7878
	201	KAPIL	17878

references

» Select * from emp-depart;

Foreign key	Dcode	Ecode
	1	101
	2	201

Foreign key

However, the above 3 tables solution is still incorrect because duplicate values can be stored in the emp-depart table.

Note :- we can take different name for parent and child columns.

Ques:- Display fees of all course.

» Select coursefees from course ;

Ques:- Display total fees of all courses

» Select Sum (coursefees) from course ;

Ques:- Display Dept wise total course fee of all courses

→ wrong Method :-

» Select sum (coursefees) from course where dept = 'MCA' ;

OUTPUT : Sum (coursefees)

125000

The drawback of above method is that it shows course fees of only one department and also you have to specify the correct spelling of department. To avoid this method and to create automatic groups group by clause is used.

» select sum (coursefees) from course group by dept ;

When we use group by in any query only following two options can be written after select clause :-

(i) Any function

(ii) AND/OR the name of the column on the basis of which groups have been made.

» Select course, sum (coursefees) from course group by dept ;

OUTPUT : Error

» Select * from emp ;

OUTPUT :

Ecode	Ename	Mgr
1	Amit	null
2	Kapil	null
3	Mudit	1
4	Rohit	6

Error :- violated parent key not found

Note : A foreign key column can accept

→ Duplicate values

→ Null values

→ Not any other value than primary key value

→ GROUPING :- Grouping is used to apply aggregate and other functions on internal sub group of data. for eg :- consider the following table :-

» Select * from course ;

OUTPUT :

CourseCode	Course	Coursefees	CourseSemester	Dept
101	MCA	80000	6	MCA
102	MBA	50000	4	MGMT
103	Btech	95000	8	ENGG
104	BCA	25000	6	MCA
105	Bscit	35000	6	ALLIED
106	Mscit	35000	6	ALLIED
107	BBA	30000	6	MGMT
108	OCP	20000	2	MCA
109	PGDM	25000	3	MGMT
110	CCNA	15000	3	ENGG

» Sub Query :- a sub query is a query which is written inside another query and its result is not displayed on screen rather the result is passed to the calling parent query.

Example :- Consider the course table as in previous topic.

Ques :- Display all course fees.

```
» select coursefees from course ;
```

Ques :- Display highest coursefees.

```
» select max(coursefees) from course ;
```

Ques :- Display details of course which is having highest fees

→ wrong methods :-

```
» select * from course where coursefees = 80000 ;
» select * from course where coursecode = 101 ;
» select * from course where coursename = 'MCA' ;
```

The above queries are giving correct answers but the method is wrong because we have to lookup highest coursefee in course table rather it should be done automatically. Therefore, we will use subquery for this.

```
» select * from course where coursefees = max(coursefees) ;
```

OUTPUT : error : Group function is not allowed here.

we cannot use a function directly on the RHS of any relational operator. Therefore, we need to replace this by a subquery.

```
» select * from course where coursefees = (select max(coursefees) from
  course ) ;
```

```
» select dept , sum(coursefees) from course group by dept ;
```

OUTPUT :

DEPT	sum(coursefees)
MCA	125000
ENGG	60000
ALLIED	75000
MGMT	105000

→ Eliminating Group Result :-

Sometimes you may want to apply certain condition on group results in that case having clause is used.

```
» select sum(coursefees) from course group by dept having
  sum(coursefees) > 100000 ;
```

```
» select sum(coursefees) from course having sum(coursefees) > 10000 ;
```

OUTPUT : sum(coursefees) — xxxx

or Error : since having can only be used with group by clause.

Note :- In Group by clause , where is used to filter rows before grouping and Having is used to exclude records after grouping.

→ Restricting the records from going into the group :-

If you want to stop a record to be a part of a group you should use where clause.

```
» select sum(coursefees) from course where coursefees > 30000
  group by dept ;
```

Here, percentage represents zero or more character

>> select * from student where city = '%%';

OUTPUT : No rows selected

Note : we don't use equal operator '=' with pattern matching operators like underscore & percentage.

The underscore pattern operator is used for exactly matching one number of character.

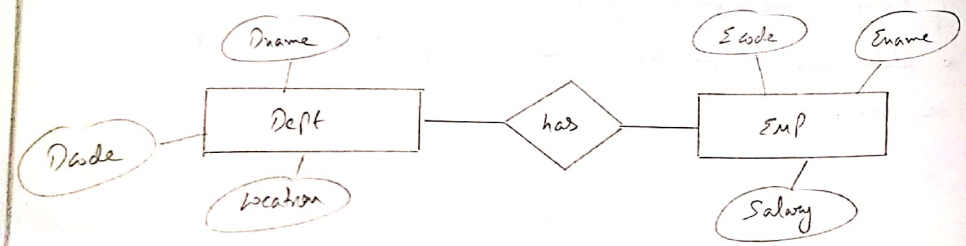
>> select * from student where upper(city) = 'KANPUR';

→ it will show details of all city where city name is in random manner i.e capital, small, initial capital, middle small, etc.

→ ANOMALIES AND NORMALIZATION :-

Normalization : In RDBMS, the tables are usually designed based on ER Diagram. Therefore, deciding number of tables is a complex technical process sometimes you can design a table that appears to be correct but it can have number of problems. Therefore, we need some parameters which we can check in a particular table to determine that it is correct or not. for example :-

Ques :- A company has a number of departments in different cities and each department has a number of employees draw a ER diagram & convert it into a table.



Ques :- Display highest coursefee

>> select max(coursefee) from course ;

Ques :- Display 2nd highest coursefee

>> select max(coursefees) from course where coursefees not in (select max(coursefees) from course); OR

>> select max(coursefees) from course where coursefees != (select max(coursefees) from course);

→ using method :-

>> select max(coursefees) from course where coursefees != 8000;

Ques :- Display 3rd highest coursefees.

>> select max(coursefees) from course where coursefees < (select max(coursefees) from course where coursefees < (select max(coursefees) from course));

Ques :- Display nth highest coursefees.

>> select coursefee from (select coursefees, dense_rank() over (order by coursefees desc) r from course) where r = &n;

>> Select coursefees from course c1 where &n = (select count (Distinct (c2.coursefees)) from course c2 where c2.coursefees >= c1.coursefees);

>> PATTERN MATCHING :- It is used to increase the scope of select and where clause.

>> Select * from student where name like 'A%';

(iv) Updation Anomaly :- Because of this anomaly there can be inconsistent values in the table.

For example : if we change the name of HR department of one record then the values will be inconsistent / invalid that spelling is incorrect.

In order to remove above anomalies Design the table based on correct ER Diagram. Therefore, if we look back at above ER diagram it clearly indicates that there should be 2 tables or 3 tables because of one to many relationship.

first table will be Department table

>> Select * from Department ;

OUTPUT:

Dcode	Dname	Location
10	Sales	Delhi
20	Marketing	Bombay
30	Production	Delhi
40	Packing	Ramper
50	HR	DLI
70	IT	Delhi

Second table will be Employee

>> select * from emp ;

OUTPUT :

EmpCode	EName	Manager	Dept
1	Kumar		40
2	Sumit	1	
3	Rohit	2	10
4	akash	1	30
5	alchil	2	10
6	tarun	3	20
25	mohan	4	
26	Rohit	5	10
99	abc	7	10
:	:	:	:

Now, Suppose we create a following table :-

Ecode	Ename	Salary	Dcode	Dname	Location
101	A	8	11	HR	Delhi
102	B	9	22	Sales	DDN
103	C	7	11	HR	Delhi
104	D	8	11	HR	Delhi

The above table appears to be correct but however it has nos of problems or anomalies.

(i) Redundancy Anomaly :- As we can see that there are three employees in dept no 11. therefore three times HR & Delhi is repeated this is redundancy & undesir

(ii) Insertion Anomaly :- Due to this anomaly the values of one type of entity are dependent on another type of entity.

For example : From the above table it looks like that there are only two departments HR & sales. However there can be many more like accountant, Marketing etc However you can not enter the name of any department until an employee joins that department. This is an Anomaly.

(iii) Deletion Anomaly :- Due to this type of Anomaly the values of one type of entity is lost because of deletion of value of another type of entity. for example :- If an employee has resigned & we need to delete his records then the record of sales department will also be lost.

However, the Cartesian Product is of no use itself because it contains many invalid or spurious. therefore, we need to filter out invalid records. So we will convert this Cartesian into joins. Joins are of many types :-

- → Self join
- → Equi join
- → Non-Equi join } Theta (θ) Join
- → Natural join
- → Outer join
  - → Left Outer JOIN
  - → Right Outer JOIN
  - → FULL Outer JOIN

» Equi JOIN :-

→ Equi join using oracle syntax

» Select * from one, two where one.name = two.name ;

OUTPUT :

Name	City	Name	HOBBY
abc	ddn	abc	cricket
abc	ddn	abc	songs
ghi	hyd	ghi	Painting
mno	ddn	mno	Painting
def	mrt	def	tracking
mno	ddn	mno	skating

→ ANSI syntax for equi join is also known as natural join

» Select * from one natural join two ;

OUTPUT : 06 rows selected [Same as above]

Note :- The natural join should be use only when both the tables have common column with same spelling and you want to match for equality.

» Select * from emp natural join department ;

OUTPUT : It will give 04 records without performing natural joins because the column name department is not same.

In such cases only equi join is the solution

» Select * from emp, department where department.deptno = emp.deptno ;

→ CARTESIAN PRODUCT & JOINS :-

Since, in RDBMS the Data is spread across multiple tables to avoid Anomalies we cannot see complete information from one table. Example :- By looking at the employee table you cannot tell the name of department. Therefore, in order to display complete information to the user we need to perform joins.

A Join is a cartesian product followed by a condition. A cartesian product is a cross join or one to one pairing of all tuples of relation 1 with all tuples of relation 2.

for example :- Consider the following two tables :-

» Select * from one ;

OUTPUT :

Name	City
abc	ddn
def	mrt
mno	ddn
ghi	hyd

» Select * from two ;

OUTPUT :

Name	HOBBY
abc	cricket
abc	songs
ghi	Painting
mno	Painting
def	tracking
mno	skating

In oracle, there are two syntaxs for performing Cartesian & joins :-

(i) ANSI SQL Compliant Syntax

(ii) Oracle Proprietary Syntax

» Cartesian Product OR Cross JOIN OR Cross Product :-

→ ANSI Syntax :-

» Select * from one cross join two ;

→ Oracle Syntax :-

» Select * from one, two ; [It joins each row of one with each row of two]

OUTPUT : 24 rows selected

>> Select emp.*, dname from emp, department where dept = dcode;

OR

>> Select empcode, name, manager, dept, dname from emp, department where dept = dcode;

>> OUTER JOIN :-

When we are displaying employee details along with department details only 09 records are displayed. however, the no. of employee are 14. so, some information is lost during Join because only matching records are displayed.

Ques: Display details of all employees along with there department details if any.

→ The Above question indication or hints the use of outer Join.

oracle syntax for outer join :- Add (+) in opposite table column

>> Select * from emp, department where dept = dcode(+);

OUTPUT: 14 rows are selected

Emp	Name	Manager	Dept	Dcode	Dname	Location

FULL TABLE — ONLY REQUIRED

In ANSI Syntax :-

>> Select * from emp left OUTER JOIN department on (dept = dcode);

>> Select * from emp RIGHT OUTER JOIN department on (dept = dcode);

>> Select * from emp, department where dept (+) = dcode;

>> Select * from emp, department where dept (+) = dcode (+);

Error : a predicate may reference only one outer joined table

>> Select * from emp FULL OUTER JOIN department on (dept = dcode);

OUTPUT : 16 rows selected

Note: In case of Equi join we can skip the name of table which is used with dot operator & column name of the name of the columns are different.

>> Select * from emp, department where dcode = dept;

Note: Results of natural join & Equi join is not same always as natural join remove common column having repeated values where as equi join will not.

>> Select * from one, two where name = name;

OUTPUT :- Error : column ambiguous defined

=> If the join condition column name is same in both the tables then you need to specify table name qualifier with dot operator.

Restricting Columns in Join :-

Ques: Display the details of all employees along with there department details.

>> Select * from emp, department where dcode = dept;

Ques: Display empcode, emp name & department name.

>> Select empcode, name, dname from emp, department where dept = dcode;

OUTPUT:

EMPCODE	NAME	DNAME
1	Kumar	Packing
3	Rohit	Sales
:	:	:

Ques: Display employee details along with department name.

Ques: Display emp code, employee name & its Manager code

» Select empcode, Name, Manager from emp;

Ques: Display emply code, employee name, its Manager Name

» Select e1.empcode, e1.name, e2.name from emp e1, emp e2 where e1.manager = e2.empcode;

» Select e1.empcode, e1.name, e2.name from emp e1, emp e2 where e1.manager = e2.emp code(+);

OUTPUT: It will show all whose manager are not assigned at all

Ques: Display total no. of departments

» Select Count(*) from department;

Ques: Display total no. of employee in each department

» Select count(*) from emp Group by dept;

» select dept, count(*) from emp Group by dept;

Ques: Display names of those employees who have not attached to any department.

» Select name from emp where Dept is NULL;

Ques: Display Names of those employees who are working in packing department.

→ wrong Method :-

» Select Name from emp where dept = 40;

OUTPUT:

Name
kumar
:

» SELF JOIN :-

Some time we need to join the table with itself to find answers of certain queries based on many relationships. However, a self join cannot be performed without table alias.

for example :

» select * from one, one;

Error : column ambiguously defined

The above query gives an error because table alias is missing

» select * one t1, one t2; [Here t1 & t2 are table alias]

OUTPUT:

Name	City	Name	City
abc	ddn	abc	ddn
abc	ddn	def	mrt
:	:	:	:

Now consider the following table,

EMPCODE	NAME	MANAGER	DEPT
1	kumar	—	40
2	Sumit	1	—
3	Rohit	2	10
4	akash	1	30
5	akhil	2	10
6	Tarun	3	20
25	Mohan	4	—
26	Rohit	5	10
99	abc	7	10
777	Rita	7	40
44	amit	5	40
55	Kapil	5	99
8	akash	7	88
66	kapil	77	—

→ FUNCTIONAL DEPENDENCY :-

Functional Dependency is an inter relationship among the attributes of a tables. In another word, an FD determines which attribute is dependent on which other attribute. An FD is always written as $f : x \to y$ where x is known as determinant & y is known as dependent.

FD $f : x \to y$ is said to exist / true / valid / implied if and only if for every single value of x, there is a single unique value of y. For example :-

Rollno	Name	City
1	A	DDN
2	B	DDN
3	A	DDN
4	C	MRT

Consider the above following table and verify the following FD :-

$f : [\text{Name} \xrightarrow{x} \text{Rollno}]$ i.e. Name can determine Rollno or not

An FD f will be valid only if :-

(i) Take t_1 & t_2 two tuples from R with same x value.

(ii) Now compare y value of t_1 & t_2 if both y value are same then the FD exists otherwise it does not hold.

Example :-

	x	y	
t_1	A	1	$t_1[y] \neq t_2[y]$
t_2	A	3	

Therefore, this FD is not valid / exist / true / implied. However, now consider following FD :-

→ Correct Method :-

>> select name from emp where dept = (select dcode from department where dname = 'Packing');

Desc: Display names of those department in which more than one employee are working.

>> select dname from department where dcode in (select dept from emp group by dept having count(*) > 1);

>> select dname from department where dcode = (select dept from emp);

Error: single-row subquery returns more than one row

Note: In subquery, 'IN' is used instead of '=' operator.

→ Normalization :- It is the process of breaking a large relation having anomalies into smaller tables such that the new tables are free from anomalies.

Normalization is a technique in which a table or a relation is tested against certain FD or functional Dependency based rules.

In normalization, we check that a table passes a normal form or not based on a FD rule that can be verified using armstrong axioms or Inference Rule.

Normalization is a process of correcting the design of a table on the basis of FD rules.

f1 : Reflexivity : ($x \to y$ and $y \subseteq x$)

f2 : Augmentation : ($x \to y$) $\models$ ($xz \to y$ and $xz \to yz$)

f3 : Transitivity : ($x \to y$ and $y \to z$) $\models$ ($x \to z$)

f4 : Additivity : ($x \to y$ and $x \to z$) $\models$ ($x \to yz$) // Union

f5 : Projectivity : ($x \to yz$) $\models$ ($x \to y$ and $x \to z$) // Decomposition

f6 : Pseudo Transitivity : ($x \to y$ and $yz \to w$) $\models$ ($xz \to w$)

[$x \xrightarrow{\text{Kumar}} y$] i.e 1, 40 → Kumar, $xz \to y$; 1, 40 → Kumar, $xz \to yz$

>> Select name from emp where empcode = 1 ; [f1 : Reflexivity]

OUTPUT :

NAME
KUMAR

>> Select name from emp where empcode = 1 and dept = 40 ;

[f2 : Augmentation]

OUTPUT :

NAME
KUMAR

>> Select name, dept from emp where empcode = 1 and dept = 40 ;

OUTPUT :

NAME	DEPT
KUMAR	40

=> f4 : Additivity

>> Select dept from emp where empcode = 1 ; [$x \to y$]

empcode → dept

>> Select name from emp where empcode = 1 ; [$x \to z$]

empcode → Name

>> Select name, dept from emp where empcode = 1 ; [$x \to yz$]

empcode → dept, Name

[Similarly, f5 : Projectivity]

FD : [Rollno → city] i.e

	x	y	
t_1	1	DDN	$t_1[y] = t_2[y]$
t_2	2	DDN	

Here, no two tuples can be found as no two same values of x are there. This FD exists.

So, we can say that rollno can determine city or in other words city is dependent on rollno for its identification.

Similarly following FD is valid :-

FD : [Rollno → Name]

However, [Name → city] is currently true for this table but if we insert one value as S, A, HRD then it will be false.

So the following FDs are present in the above tables :-

[Rollno → Name] And [Rollno → City]

And following FDs cannot be determined :-

[Name → Rollno] [Name → City]

[City → Name] [City → Rollno]

However, in case of large table it is sometimes difficult to determine certain FDs which are either hidden or implied ($\models$).

An implied FD is an functional dependency which is not directly visible but can be proved with the help of armstrong axiom or inference rules.

There are total six rules that are said to be sound & complete :-

Ques : Is the above table is in first Normal form ?

Answer : No, the above table is not in first Normal form because

(i) The column name is not atomic if it can be broken into first name, middle name and last name.

(ii) The column phone contains multiple values in a single row.

The above table can be redesigned as,

Rollno	FName	MName	LName	Ph1	Ph2	Ph3
1	Amit	Kr		484	812	
2	Rohit			418	982	316
3	Mohit	Singh	Sharma	214		

» Second Normal form : The 2NF is related to the concept of full functional Dependency (FFD). A table is said to be in 2NF if it does not have any partial dependencies. In other words, all the non prime attributes should be completely dependend on the complete primary key and not on any part of it.

for example :- Consider the following table

Rollno (Prime key)	Name (→ NP)	DOB (→ NP)
1	A	4-July-76
2	B	4-July-76
3	A	8-oct-78

In the above table, Rollno is atomic primary key i.e it is made up of only single attribute. So, rollno is called as Prime attribute And

Ques :- Given a relation R(ABCDEF) and set of FDs :-

f : A → BC, B → E, CD → EF

Determine that AD → f.

Answer : Given, A → BC

By using f5 : A → B, A → C

By using f2 : AD → CD

Given, CD → EF

By using f3 : AD → EF

AD → E, AD → F [Hence, Proved]

→ NORMAL FORMS :- A Normal form is a FD based condition satisfied by a relation (table). There are many types of Normal forms :-

→ 1NF → 3NF → 4NF
→ 2NF → BCNF → 5NF

Although these Normal form are not sequential. Any normal form can be tested directly and it tells the health of the table i.e anomalies are present or not.

» first Normal form : The 1NF is related to the concept of atomic columns & singular values. for example :- Consider the following table

Rollno	Name	Phone
1	Amit Kr	484, 812
2	Rohit	418, 982, 316
3	Mohit Singh Sharma	214

Only Prime not Pkey

Item No → Item Name (NP)

Partial FD → undesirable

Because of this partial FD the above table is not in 2 NF and it is an example of bad design so we need to normalise it.

→ Normalising a Non-2NF into 2NF :-

Step (01) Identify partial FD of the form X → Y

Step (02) Remove Y from main table

Step (03) Create a new table XY with X as primary key.

For example :- Lets consider the above item table / order table.

Now apply the above steps on the unorganized order table :-

Step (01) Item No → Item Name

X → Y

Step (02) Remove Item Name

Step (03) Creating a new table Items.

Item No (X, Pkey)	Item Name (Y)	Item Price
1777	Colgate	75
2888	Pepsi	65
3999	Lux	45
5555	Fa	35

So, the required final two tables will be :-

The remaining two attributes name & DOB are FFD on complete primary key and not on any part of primary key. Therefore, the following two FD exist :-

(i) Rollno →(FFD) Name (NP)

(ii) Rollno →(FFD) DOB (NP)

So the above table is in 2 NF. However, consider the following table.

Prime — Composite key — Prime

Order No	Item No	Item Name (→NP)	Item Price (→NP)	Qty (→NP)
111	1777	Colgate	75	2
111	2888	Pepsi	65	11
111	3999	Fa	35	2
222	1777	Colgate	75	12
222	5555	Lux	45	11
333	2888	Pepsi	65	12

In the above table, order no, item no is composite key. Therefore, following FDs are present :-

Pkey { orderno Item no → Item Name; order no Item no → Item Price; order no Item no → Item Qty } NP

However, the above table design is not correct and it has number of anomalies. It is because there are some partial FD's.

for example :- whenever you try to find the name of Item no 2888 it comes out to be Pepsi.

i.e., Year → Hostle
X → Y
NP → NP

This FD can also be written as,

Rollno → Year , Year → Hostle ∴ Rollno → Hostle

So, the above table is not in 3NF because of the transitive FD.

For Example :- whenever the year is 1st the name Ganga is unnecessarily repeated. however, the name of 4th year, 5th year Hostle are not known.

So, now we will Normalise / Decompose this table. During decomposition you must make sure that the decomposition is :-

(i) Dependency Preserving (ii) Loss less

Step (01) Identify the NP to NP FD of the form X → Y

Here, Year → Hostle
X → Y or NP → NP

Step (02) Remove Y from main table

Step (03) create a new table XY with X as Primary key

(X) Year (PKey)	Hostle (Y)
1	Ganga
2	Sai
3	Satya
4	Shubash
5	Yamuna

(I)

Item No (PKey)	Item Name	Item Price
1777	Colgate	75
2888	Pepsi	65
3999	Fa	35
5555	Lux	45

(II)

Order No (Composite Key)	Item No (Composite Key)	Qty
111	1777	2
111	2888	11
111	3999	2
222	1777	12
222	5555	11
333	2888	12

» Third Normal Form :- The 3NF is related to the concept of transitive FDs, a table is said to be in 3NF if it does not have any transitive FDs or in other words, it does not have any non-prime to non prime FD i.e NP to NP FD i.e NP → NP.

for eg :- Consider the following table :-

Rollno	Name	Year	Hostle
1	A	1	Ganga
2	B	1	Ganga
3	C	2	Sai
4	D	2	Sai
5	E	1	Ganga
6	A	1	Ganga
7	B	3	satya

In the above table, Rollno is atomic Primary Key therefore following FDs are present :-

Rollno → Name Rollno → Year Rollno → Hostle

All the three NP are FFD on complete Primary key Rollno. Therefore, the above table is in 2NF but still the table is not correct because the non prime attribute Hostle is dependent on Non prime Year.

The above table design is still not correct because suppose if a student has 5 phone numbers. The above problem is coming because we are trying to keep SVD and MVD in the same table.

Rollno → Name (SVD)
Rollno → DOB (SVD)
Rollno →→ Phno (MVD)

So, We will now normalise the above table :-

Step (01) Identify the MVD of the form $X \twoheadrightarrow Y$

Rollno →→ Phno
Pkey →→ NP
$X \twoheadrightarrow Y$

Step (02) Remove Y from main table

Step (03) Create a new table XY as composite key

Composite key

Rollno	Phonono
1	318
1	418
1	777
4	818
4	772

So, the final two tables are :-

Rollno	Phone NO
1	318
1	418
1	777
4	818
4	772

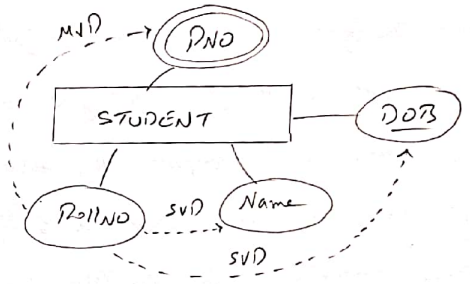
Rollno	Name	DOB
1	A	4 oct
2	B	8 Dec
3	A	17 Jan
4	D	4 oct

Read Yourself
1D
BCNF
5NF

>> Fourth Normal form :- The 4NF is related to the concept of Multivalues Dependency (MVD) i.e, $X \twoheadrightarrow Y$

A Table is said to be in 4NF if SVD (single value Dependency) and MVDs are not in the same table.

So the table should contain either SVD or only MVD.



Now suppose following table is created :-

Rollno	Name	DOB	Pho
1	A	4 oct	418, 3814, 315
2	B	8 Dec	
3	A	17 Jan	746, 666

The above table design is not correct because it violates 1NF.

Therefore, the table is redesign as :-

Rollno	Name	DOB	Ph1	Ph2	Ph3
1	A	4oct	318	418	219
2	B	8 Dec			
3	A	17 Jan	888	666	
4	D	4 oct			