Modularity : Breaking a big program into smaller subprograms(functions)

Advantages of modularity

1. Easy to understand small program
2. Easy to debug small program
3. Easy to maintain small program
4. Reusability
5. Removes duplicacy

Normalization : Divide a big table into smaller tables

In Normalization : Functional Dependency

It tells us which columns are dependent on which other columns

It tells us that which columns determines which other columns

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rn | Code | Name | Marks | Address |
| 1 | E1 | Ajay | 67 | Delhi |
| 2 | E2 | Deepak | 89 | Delhi |
| 3 | E3 | Manoj | 86 | Calcutta |
| 4 | E4 | Sagar | 76 | Delhi |

Candidate Key : Column which contains properties of becoming a primary key

Rn , Code : (Candidate Key)

Rn : Primary Key

Code : Alternate Key (Leftover key out of candidate key becomes alternate key)

FDependecy > Rn > Name , Code , Address , Marks

1 NF : **Every cell should have atomic values**

2 NF

3 NF

Joins : It is used to fetch records from more than 1 table

Different types of Joins

* Inner Join : It gives you matching records
* Outer Join : It gives you matching as well as unmatching records

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

* Cross Join
* Self Join

(For inner join & outer join, there has to be a common column)

drop table student

create table student(rn int primary key,

name varchar(20), address varchar(20))

create table marks(rn int foreign key references student(rn),

science int, hindi int , eng int)

insert into student values

(1,'Ajay','Delhi'),

(2,'Vijay','NDelhi'),

(3,'Prem','ODelhi'),

(4,'Sagar','Calcuta'),

(5,'Deepak','Delhi'),

(6,'Garv','Delhi')

insert into marks values

(1,23,34,45),

(5,23,34,45),

(4,23,34,45)

select \* from student

select \* from marks

-- Inner Join

select name , science , eng, hindi from

student join marks

on student.rn = marks.rn

select student.name , marks.science , marks.eng, marks.hindi from

student join marks

on student.rn = marks.rn

select x.name , y.science , y.eng, y.hindi from

student x join marks y

on x.rn = y.rn

-- Outer Join

select x.name , y.science , y.eng, y.hindi from

student x left outer join marks y

on x.rn = y.rn

select x.name , y.science , y.eng, y.hindi from

student x right outer join marks y

on x.rn = y.rn

select x.name , y.science , y.eng, y.hindi from

student x full outer join marks y

on x.rn = y.rn

-- Cross Join : Cartesian Product

-- There is no need to have a common column

create table courses(coursename varchar(20), duration int)

insert into courses values ('C', 90), ('C', 90),

('C++', 20),

('C#', 67),

('Java', 190)

-- All the students have to be enrolled in all courses

Select student.\* , courses.\* from student cross join courses

-- In Cross Joins, columns are added , records are multiplied

-- Self Join : When a table is joined to iteself

select \* from employee

select \* From sys.tables

select \* from department

-- List Employee names & their manager names

select x.name , y.name from employee x join employee y

on x.managerid = y.id

-- SubQueries

-- Query inside other Query

-- Outer Query OPERATOR (Inner Query)

-- OPERATOR in case inner query returns single value ,

-- = <> > < >= <=

-- OPERATOR in case inner query returns more than single value ,

-- IN , > any , > all

select \* from employee

-- Give employee name who gets max salary

select name from employee where salary = (select max(salary) from employee)

select name from employee where salary > (select salary from employee

where name='Deepak')

select name from employee where salary > all (select salary from employee

where state IN ('UP', 'Delhi'))