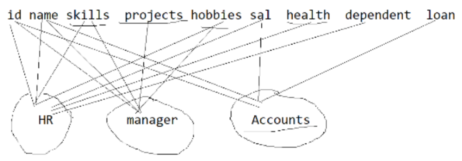
Views



When you want to provide limited access to the existing data, then we create views

views are of 2 types

1. views
   a. for view no separate memory is allocated for storing data,
   b. only base query will be stored, and any statement on the view will use base query to get the data, because of that we always get UpToDate data in view
   c. if view contains all not null columns of the single base table and if it is not read only view, then we can use all DML operation(insert, delete, update) on the view
   d. if the view is based on joins, aggregate functions, group by statement or union of multiple queries, then by default the views are readonly

```
select * from emp e
where sal<(select avg(sal) from emp m where m.mgr=e.mgr)

create a view fac_room
as
select cid,cname,fid,fname,null,null
from course c right join faculty f on f.fid=c.fid
where c.cname is null
union
select cid,cname,null,null,rid,rname
from course c right join room r on r.rid=c.rid
where c.cname is null
```

uses of views:
1. Hide complexity of the queries (joins, aggregate functions nested queries)
2. To give restricted access to few columns or rows from tables
3. Hide table names, to increase the security of data.

2. Materialized view

a. Views for which the first time the base query will get executed and then the output will be stored in a temporary table in RAM, within the session, the data will be retrieved from the RAM
b. you may not get uptodate data in materialized view.

```
 create materialized view myview

as

select * from emp;
```

to drop view

```
drop view myview
```

to get the 3rd highest salary

```
select *
   -> from emp s
   ->  where 2=(select count(*)
   ->  from (select distinct sal from emp) e
   ->  where e.sal>s.sal)
   -> order by sal;
```

to get 3 topmost salaried employee

```
select *
   -> from emp s
   ->  where 3>select count(*)
   ->  from (select distinct sal from emp) e
   ->  where e.sal>s.sal)
   -> order by sal;
```

to get 3 bottommost salaried employee

```
select *
   -> from emp s
   ->  where 3>select count(*)
   ->  from (select distinct sal from emp) e
   ->  where e.sal<s.sal)
```

-> order by sal;

to create index

create index sal_idx

om emp(sal)

to drop index

drop index sal_idx

to list all indexes available on the table

SHOW INDEXES FROM customers

to find which indexes are used by the query or to suggest using of which indexes in query

```
EXPLAIN SELECT *
FROM
  customers
WHERE
  contactFirstName LIKE 'A%'
    OR contactLastName LIKE 'A%';
```

MySQL Engine INNODB supports ACID property

A—automicity-→ every transaction will get executed as a single unit

```
begin transaction
    withdraw amt form source
    deposit amt in the dsestination
end transaction
```

c—consistency---correctness of data will be there after every transaction completes

I ----isolation---- every transaction will get executed in isolation,

intermediate changes are visible only to the user who is performing the transaction,

these changes will visible to other users, when the transaction is commited

D- durability------It will show the performance for longer period of time

Transaction control language

commit, rollback, savepoint

emp

10 records

commit

insert 1

insert 1

update 1

commit

delete 1

rollback

10 records

commit

insert-3

update 2

savepoint A

delete 1

insert-3

update 2

savepoint B

insert-2

delete 1

rollback to B

12 rows

insert -3

delete 1

create table

insert- 4

delete 1

rollback