

## indexes in mysql

Indexes are of 2 types

1. clustered index →
  - a. there is only one clustered index,
  - b. it is stored along with table data,
  - c. it does not require extra space to store index file
  - d. the data is stored in the sorted order based on clustered index
2. non clustered index →
  - it is stored outside the table,
  - there can be any number of non clustered index files,
  - every index file stores, key and the position in the table

Types of indexes in mysql

1. primary key index- when we add primary key constraint in the table, this index is automatically created, data in the table is sorted based on this index
2. unique index- When you add unique constraint in the table, unique index is automatically gets creates, it does not allow to add duplicate not null values in the column.  
if unique constraint is not assigned in the table, but if you create unique index then also duplicate values will not be allowed in the column
3. full text index---usually it is used on tex/tinytex/mediumtext/longtext columns, in which we need to search jargons, then use full text indexes, usually these are used in search engines.
4. regular index---if you create a index on a table, then it is regular index
5. spatial index---- if the geolocationof data is stored in the table, then we use spatial indexes.
6. descending index---while creating indexes if you use desc keyword then those indexes are called as descending indexes.

## to create index

```
create index sal_idx  
on 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%';

write a command to create descending index on job, if job is same then arrange the data on sal

create index idx\_job

on emp(job desc,sal)

MySQL Engine INNODB supports ACID property

A—atomicity-→ 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 committed

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

Transaction control language

commit, rollback, savepoint

emp

10 records

commit

insert 2

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

### window functions

row_number()	<p>It assigns a unique sequential number to each row, starting with 1, according to the ordering of rows within the window partition.</p> <p>select empno,ename,sal,row_number() over (order by sal desc) rownum from emp e ----- it wil generate only one window in table and assign numbers 1,2,3,4.....</p> <p>select empno,ename,sal,row_number() over (partition by deptno order by sal desc) rownum from emp e ----- it wil generate one window for each department in table and assign numbers 1,2,3,4..... in each window</p>
rank()	<p>The <code>rank()</code> function will assign the same rank to the same values i.e. which are not distinguishable by ORDER BY. Also, the next different rank will not start from immediately next number but there will be a gap i.e. if 4th and 5th employees have the same salary then they will have the same rank 4, and 6th employee which has a different salary will have a new rank 6.</p>

dense_rank()	The dense_rank function is similar to the rank() window function i.e. same values will be assigned the same rank, but the next different value will have a rank which is just one more than the previous rank, i.e. if 4th and 5th employee has the same salary then they will have the same rank but 6th employee, which has different salary will have rank 5,
--------------	--

row\_number, rank, dense\_rank, lag, lead, first\_row

to find highly paid employee in each department

select \*

from (SELECT empno,ename,sal,deptno,row\_number()

over( partition by deptno ORDER BY sal DESC) AS rn,

dense\_rank() over (partition by deptno order by sal desc

) drn from emp) e

where e.drnr=1;

lead function will give you the next value within frame, lag function give you previous value with frame.

select empno,ename,sal,lead(sal,1) over (order by sal) leaddata, lag(sal,2) over (order by sal) lagdata,first\_value(sal) over (order by sal)

-> from emp;

---

PL\_SQL—(Procedural language -Structured Query Language)

procedure	any block of code , which has business logic is called as procedure
function	any block of code , which has business logic, and returns one values, is called as functions. these can be used in select clause and where clause in SQL
triggers	any block of code , which has business logic, and gets called on some users action automatically, then it is called as trigger
exception	While executing procedures if any error occurs, we use exceptions
cursors	When you want to traverse through all rows one by one, and perform some action on each row, on by one, then use cursors

Types of parameters to the procedure

in	these are read only parameters. these are default parameters. these are used to send the input value to the procedure
out	these are write only parameters these are used to get the output from the procedure

inout	these are both read and write parameters, we can send the value to the procedure, and inside procedure we may change the value of the parameter
-------	---

Why we use PL SQL

1. we can hide table names from the developer of the middleware application, which increases the security of the database.
2. For a particular task, if we need to execute many queries, then we may wrap these queries in a procedure, and call the procedure from middleware application, once, execute all the queries, complete the task and go back, this will reduce the network traffic, also improves performance efficiency of the middleware application. so it reduces the interaction between middleware program and database.
3. If any of the query is complex, then we may hide the query inside the procedure
4. Procedures will also reduce the network traffic.

delimiter //

create procedure <procedure name>(parameters...)

begin

declaration of variable;

statement1;

statement2;

end//

delimiter ;

call <procedurename>();

1. to insert record into dept table

delimiter //

create procedure insertdept(in did int,dnm varchar(20),dloc varchar(20))

begin

insert into dept values(did,dnm,dloc);

end//

delimiter ;

call insertdept(10,'admin','pune')

2. write a procedure to accept eid, sal and job from user as i/p and update sal and job of the employee in emp table

delimiter //

create procedure updateemp(eid int,esal float(9,2),ejob varchar(20))

begin

update emp

set sal=esal,job=ejob

where empno=eid;

end//

delimiter ;

```
call updateemp(7902,6666,'QA');
```

3. write a procedure finddata, to get sal and comm of the employee  
delimiter //  
create procedure findjob(eid int,out esal float(9,2),out ecomm float(9,2))  
begin  
    select sal,comm into esal,ecomm  
    from emp  
    where empno=eid;  
end//

```
delimiter ;
```

```
call findjob(7902,@s,@c)  
select @s,@c
```

in above example, select ... into statement can be used only inside pl sql blocks, the select query should return single row as output. number of column names before into and number of variables after into should be same.  
@s and @c are session variables. these variables will remain available till the time you logout.

4. write a procedure to find number of employees and maximum netsalary for the given department.  
net sal =sal+comm  
delimiter //  
create procedure findemp(in edid int, out cnt int,out maxsal float(9,2))  
begin  
    select count(\*),max(sal+ifnull(comm,0)) into cnt,maxsal  
    from emp  
    where deptno=edid;  
end//  
delimiter ;

```
call findemp(10,@c,@ms)
```

5. write a procedure which will accept a number and increment a number by 10  
delimiter //  
create procedure incrementnum(inout cnt int)  
begin  
    set cnt=cnt+10;  
select cnt;  
end//  
delimiter ;  
  
set @c=5  
call incrementnum(@c)  
select @c;

6. write a procedure to display all employees in given department and sal >1500.

```

delimiter //
create procedure getempdata(in edid int,in esal float(9,2))
begin
select *
from emp
where deptno=edid and sal>esal;
end//
delimiter ;

call getempdata(10,1500);

```

7. write a procedure to find all employees along with dname with sal>2000

```

delimiter //
create procedure findempdetails(esal float(9,2))
begin
    select empno,ename,sal,e.deptno,dname
    from emp e,dept d
    where e.deptno=d.deptno and sal > esal;
end//
delimiter ;

call findempdetails(2000);

```

8. display feedback based on comm

```

if comm is null or 0 then display "poor performance"
if comm <=300 then display 'ok performance'
if com >301 and <=500 then display good performance
else display excellent performance.

```

<pre> if condition then     statements; else     statements end if; </pre>	<pre> if condition then     statements; elseif condition then     statements else     statements end if; </pre>
--	---

```

delimiter //
create procedure getRemark(eid int,out remark varchar(50))
begin
    declare vcomm float(9,2) default 0;
    select comm into vcomm
    from emp
    where empno=eid;
    if vcomm is null or vcomm=0 then
        set remark='poor performance';
    elseif vcomm<= 300 then
        set remark='ok performance';

```

```

elseif vcomm<= 500 then
    set remark='good performance';
else
    set remark='excellent performance';
end if;
end//

```

```

end//
delimiter ;

```

9. write a procedure to find netsal of the given employee and find the remark, if  
 netsal <1000 “less”  
 if >=1000 and <2000 then ‘ok’  
 if netsal >=2000 and < 3000 then ‘good’  
 otherwise better  
 display remark inside the procedure

```

netsal= sal+comm

```

```

delimiter //
create procedure findNetsal(eid int,out remark varchar(50))
begin
    declare vsal,vcomm,vnetsal float(9,2);
    select sal,comm into vsal,vcomm
    from emp
    where empno=eid;

    set vnetsal=vsal+ifnull(vcomm,0);

    if vnetsal<1000 then
        set remark ='less';

    elseif vnetsal<2000 then
        set remark='ok';

    elseif vnetsal<3000 then
        set remark='good';

    else
        set remark='better';

    end if;
    select eid,vsal,vcomm,vnetsal,remark;
end//

delimiter ;

```



10. write a procedure getdiscount to find discount % and discounted amount from product table for the given product

if price < 50 then 3%

if price >=50 and <80 7%

if price >=80 and < 100 8%

otherwise 12%

display pid,pname,price,discount percentage and discount amount

pid	pname	qty	price
1	lays	34	50.00
2	prigles	50	150.00
3	nice	53	50.00
4	maggi	45	60.00

delimiter //

create procedure getdiscount(dpid int, out discount float(4,2))

begin

declare vpname varchar(20) default '';

declare vprice float(9,2);

select pname, price into vpname,vprice

from product

where pid=dpid;

if vprice<50 then

    set discount=0.03;

elseif vprice<80 then

    set discount=0.07;

elseif vprice <100 then

    set discount=0.08;

else

    set discount=0.12;

end if;

select dpid , vpname,vprice,vprice-(vprice\*discount),discount;

end//

delimiter ;

In PLSQL there are 3 loops

While expression do Statements End while;	This is top tested loop, will repeat statements till the condition is true
REPEAT statements; UNTIL expression END REPEAT	This is bottom tested loop, will repeat statements until the given condition is false
Label1:Loop If condition then Leave Label1 End if endloop	<p>This is infinite loop , will continue execution till leave statement gets executed, leave statement is same as break statement, it forcefully stops the loop.</p> <p>In this loop you may use iterate statement, it is similar to continue statement in java, It will transfer the control to the beginning of the loop.</p>

1. Write a procedure which accepts start and stop values and display all numbers between start and stop
2. Example displaydata(10,20) o/p 10,11,12,13,14,15.....20

Delimiter //

Delimiter //

Create procedure displaydata(in start int,stop int)

Begin

Declare cnt int;

Declare str varchar(100) default '';

Set cnt=start;

While cnt<=stop do

set str=concat(str,cnt,,);

Set cnt=cnt+1;

End while;

set str=substr(str,1,length(str)-1);

Select str;

End//

Delimiter ;Delimiter ;

3. Write a procedure to accept a number from user and display its factorial  
Delimiter //  
Create procedure displayfactorial(in num int,out fact int)  
Begin

```

    Declare start int default 1;
    Set fact=1;
    While start<=num do
        Set fact=fact*start;
        Set start=start+1;
    End while;
End//

```

### Using repeat until loop

1. Write a procedure which accepts start and stop values and display all numbers between start and stop(use repeat ...until loop)

Delimiter //

Create procedure displaydatarepeat(in start int, in stop int)

Begin

```

    Declare cnt int default start;
    Declare str varchar(100) default "";
    Repeat
        Set str=concat(str,cnt,",");
        Set cnt=cnt+1;
    Until cnt > stop

```

End repeat;

Set str=substr(str,1,length(str)-1);

Select str;

End//

Delimiter ;

2. Write a procedure to find factorial of a number(repeat until)

Delimiter //

Create procedure displayfactorialrepeat(in num int, out fact int)

Begin

```

    Declare start int default 1;
    Set fact=1;
    Repeat
        Set fact=fact*start;
        Set start=start+1;
    Until start>num

```

End repeat;

Select fact;

End//

Delimiter ;

### Loop ...endloop

3. Write a procedure which accepts start and stop values and display all numbers between start and stop(use loop ...end loop)

Delimiter //

Create procedure displaydataloop(in start int,in stop int)

Begin

Declare str varchar(100) default '';

Declare cnt int default start;

L1:Loop

Set str=concat (str,cnt,'');

Set cnt=cnt+1;

If cnt>stop then

Leave l1;

End if;

End loop;

Set str=substr(str,1,length(str)-1);

Select str;

End//

Delimiter ;

4. Write a procedure to find factorial of a number using loop ...end loop;

Delimiter //

Create procedure displayfactorialloop(in num int,out fact int)

Begin

Declare start int default 1;

Set fact=1;

L1:loop

Set fact=fact\*start;

Set start=start+1;

If start >num then

Leave l1;

End if;

End loop

Select fact;

End//

## Cursors

Cursors are used to read the data from the table row by row, and process it

Step by step procedure to use cursor

1. Declare cursor.
2. declare continue handler to stop the loop
3. open the cursor.
4. fetch the row from the cursor.
5. check whether reached to last row leave the loop
6. process the row.
7. goto step 4
8. once come out of the loop then close the cursor.

elimiter //

```
create procedure displayallemp()
begin
    declare vset,vempno int default 0;
    declare vname varchar(20);
    declare empcur cursor for select empno,ename from emp;
    declare continue handler for NOT FOUND set vset=1;
    open empcur;
    lable1: loop
        fetch empcur into vempno,vname;
        if vset=1 then
            leave lable1;
        end if;
        select vempno,vname;
    end loop;
    close empcur;
end//
delimiter ;
```

---