L:8 D:73

DATE 28 10 2017

Pseudo Columns

- Pseudo columns are fake columns.
- Pseudo columns are virtual columns
- Pesudo column is not a column of a table.

eg: computed column (sal * 12) as "ANNUAL"

- it is a not a column of Emp table.
- it is a pseudo column.
- ANNUAL is not stored in HD.
- it is computed whenever required.

eg: expressions (sal + comm) as "TOTAL"

eg: function-based column (avg(sal)) as "AVG_SAL"

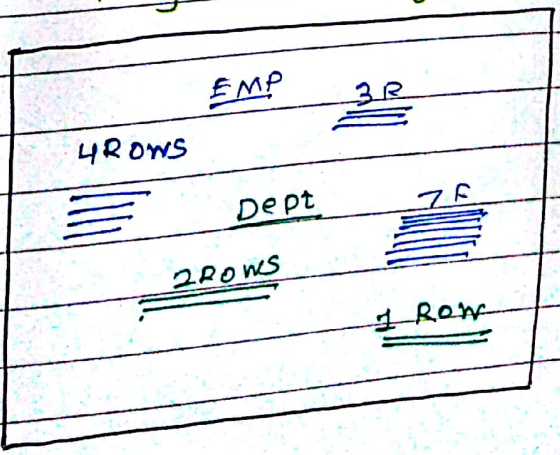
Mentioned above are the user defined pseudo columns. There are some System supplied pseudo columns.

```
Select ename, sal from emp;
```

o/p

ENAME	SAL
Smith	8000
Allen	1600
:	:

- it does not mean, Smith was inserted first in the table.
- While inserting, rows are not stored together. They are fragmented in server HDD.



PAGE

DATE

In DBMS, All the rows are stored together.
∵ All the rows are stored in ONE FILE
In RDBMS, table is not a file. every row is a file. ∴ Each row is stored separately

Why?
To make insert faster!
When you write an INSERT, it will insert row, wherever the space is available.

∴ Insert is not done sequentially it does it randomly
But, SELECT, will execute sequentially.
∴ O/p of SELECT depends on Row ~~update~~ address

If we updat 'Smith' ~~tho~~ to 'Smythe',
i.e. the row size will ~~cho~~ increase by 1byte.
If the current row location can accomodate 1byte, the changes are made at that location only. But if it cant, the row will be relocated to a diff location with more capacity location. ∴ row add of 'Smythe' will change.
∴ if we write SELECT now, 'SMYTHE' will not be the first row ~~of~~ of o/p.

only for changes in varchar type fields.

PAGE

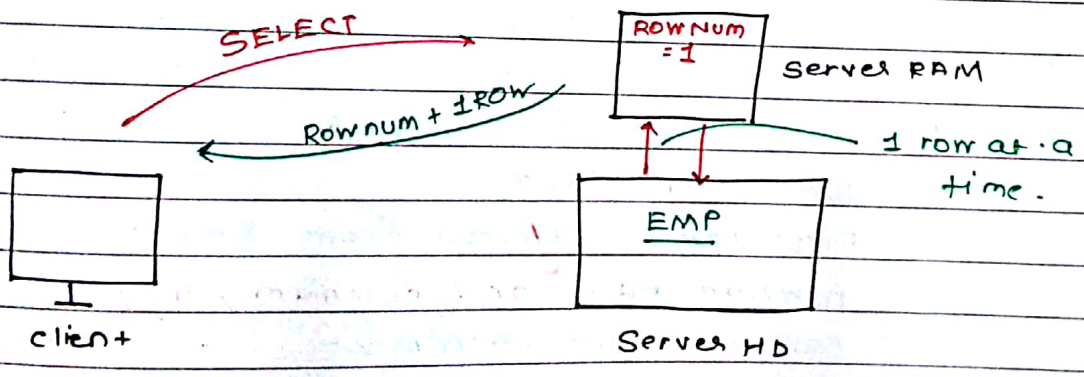
DATE

Select rownum, ename, sal from emp;

↳ o/p

ROWNUM	ENAME	SAL
1	SMITH	800
2	ALLEN	1600
:	:	:
:	:	:

- rownum is not a column.
- It is a var. ~ to count.
- It starts by 1
- It increments by 1
- It is a global variable.
- After SELECT is over, it is destroyed.
- Upper limit of ROWNUM is $9.9 * 10^{125}$



Select rownum, ename, sal from emp where rownum=1;

↳ o/p

ROWNUM	ENAME	SAL
1	SMITH	800

↳ Does not mean smith was inserted first

(No PRACTICAL USE)

PAGE

classmate

Select rownum, ename, sal from emp;
Where rownum < 4

o/p

rownum	ename	sal
1	⋮	⋮
2		
3		

Sel

Select rownum, ename, sal from emp where rownum = 4;

No rows selected

Wont work.

Select rownum, ename, sal from emp where rownum > 4;

No rows selected.

- SELECT will go to server.
- rownum = 1. (RAM)
- First row is selected from Emp from HD.
- rownum ≠ 4 or (rownum > 4);
- rownum wont increment
- rownum will stuck at 1.
- ∴ the WHERE condition will never satisfy.
- ∴ No rows.

select rownum, ename, sal from emp order by ename;

o/p

rownum	ename	sal
11	ADAM	
2	ALLEN	
14	BLAKE	

Select rownum, ename, sal from
(select ename, sal from emp order by ename)

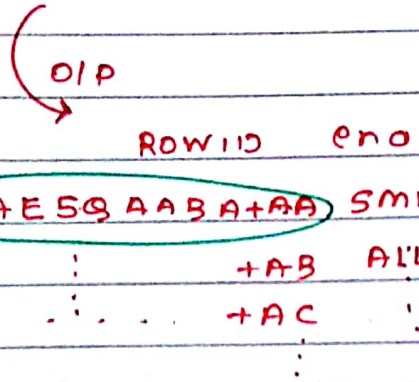
o/p

ROWNUM	ENAM	SAL
1	ADAM	
2	ALLEN	
3	BLAKE	

ROWID

- Also a pseudo column
- System Supplied pseudo column.
- ROWID → Row Identifier.

Select rowid, ename, sal from emp;



- Rowid is the Row address!
- Rowid is the address of the row in the database Server Hard Disk.
- This is the actual ~~memom~~ physical memory location where the row is stored
- It is a fixed length (18 char) char string not Hexa.
- It is an encrypted string.
- only RDBMS can decrypt it.
- Even OS would not decrypt it.
- So, when you select from table, the order of rows in the o/p will be in ascending order of ROWID.

If you update a ROW ('SMITH' to 'SMYTHE'), the row add (ROWID) may change (as explained before)

(only in case of Varchar.
∵ (Varchar is Var. length DT)

DATE

Select rowid, ename, sal from emp where rowid = '.... AAAE5gAABA+AA';

o/p

ROWID	ENAME	SAL
.... AAAE... +AA	SMITH	800

- useless query
- It is working now. but if we make changes to smith & if it changes it rowid, it wont work

where it is used practically?

It is used to update or DELETE the duplicate rows. (inserted by mistake)

ROWID	ENAME	SAL
...AAAE...+AA	SMITH	800
...AAAE...+AB	SMITH	800
	:	:

if you want to delete one & keep the other.

- You cant delete with ROWNUM.

- ROWID is internally used by RDBMS to distinguish betn 2 rows.
- No two rows of any table of entire DB can have the same ROWID
- ROWID works as a unique identifier in the whole database.
- ROWID is internally used for ROWLocking.
- The locked row's ROWID be comes read only for other users.

PAGE

classmate

DATE

- Used to manage indexes
- Used to manage cursors
- for row management.
- every row is a file
- ∴ rowid is a filename.

For programmers, there's only one practical use of ROWID:

- To delete / update duplicate rows.

Feature or the concept of ROWID is available in Oracle and MySQL and all other RDBMS.

BUT

YOU CAN VIEW ROWID ONLY IN ORACLE but NOT IN MYSQL.

∴ We cant Update / Delete base by ROWID in MySQ

- ROWNUM is available in ORACLE RDBMS and we can view it in Oracle.
- ROWNUM is NOT AVAILABLE IN MySQL

DATE

MySQL - SQL set Operators.

- Set operators are based on Set Theory.

EMP 1

EMPNO	ENAME
1	A
2	B
3	C

EMP 2

EMPNO	ENAME
1	A
2	B
4	D
5	E

Two tables with identical structure

```
Select empno, ename from emp1 .
        union
Select empno, ename from emp2;
```

- Foundation of RDBMS and SQL is based on Set Theory.
- IBM is a founder of SQL.
- Dr. Codd (Dr. E.F Codd) is the founder of RDBMS (1968). → was a Mathematician/statistician.
  → introduced the concept of Table.
- When one table exceeds its max. row capacity, an identical structured table is created that would store further rows. This table is called 'Extension table'. union can be used to combine the rows of old table & the extension table. (MySQL allows 64 TB of data per table)

classmate

PAGE

DATE

Acc. to Dr codd

- we have rows &
- we have columns.

& the intersection of row & column is DATA.

- Table is not a file, every row is a file → was Dr Codd's idea.
- Dr codd introduced the idea of Datatypes
- Dr codd introduced the idea of System table.
- Idea behind the SQL was a was Dr. Codds idea.
- Dr. codd was a consultant for IBM
- Concept of foreign key, insert, update, delete, distributed database was his idea.
- Concept of client-server arch.
- Concept of Backend & front end.
- Concept of laptops.
- Concept of Touch Screen
- Concept of mouse. (implemented by Apple)
- Larry Elison was a student of Dr. codd.

```
Select empno, ename from emp1
        union
Select  empno, ename  from emp2 ;
```

It will combine the o/p of both select's & will suppress the duplicates.

o/p

EMPNO	ENAME
1	A
2	B
3	L
4	D
5	E

DATE

- Structure of both the select statements has to match.
- Tables can have diff struct.
- Diff column names are allowed.

↳ eg:

```
Select empno1, ename from emp1
                union
Select empno2, ename from emp2;
```

↳ O/p.

Column name of first SELECT.

EMPNO1	ENAME
1	A
2	B
3	C
4	D
5	E

How Duplicates are suppressed?

- When we are using union, sorting will take place in server.

DATE

If you want to use order by, use it at the end.

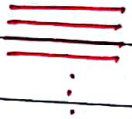
```
select empno1, ename from emp1
        union
Select empno2, ename from emp2
    order by 1;
```

(order by 1st column (empno)

- Union is used in Full Outer join.
- To combine data of employees from diff branches
- If the data is large, the data is stored in multiple tables

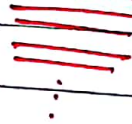
eg:

ORDERS-2016



e.g:
~~How~~ who all customers have placed orders in 2016 & 2017

ORDERS-2017



Why?

- logical div.
- MySQL allows 64 TB per table.

union all

union all will combine the o/p of both the SELECTs and the duplicates are not suppressed.

```
Select empno1, ename from emp1
        union all
Select empno2, enamr from emp2
        order by 1;
```

↳ o/p

EMPNO1	ENAME
1	A
1	A
2	B
2	B
3	C
4	D
5	E

∴ Duplicates are not suppressed. No internal sorting will take place.

Intersect

intersect will return what is common in both the selects AND the dupplicates are suppressed. (∵ Sorting).

```
Select empno1, ename from emp1
        intersect
Select empno2, ename from emp2
        Order by 1;
```

o/p

EMPNO1	ENAME
1	A
2	B

e.g
Who all customers have placed orders in 2016 AND 2017 as well

Minus

1st - 2nd

Minus will return what is present in first SELECT & not present in second SELECT AND the deplicates are suppressed (∵ intesnal sorting)

```
Select empno1, ename from emp1
        minus
Select empno2, ename from emp2
        order by 1;
```

o/p

EMPNO1	ENAME
3	C

e.g
Who all customers have placed

Not

Exp is Top to Bottom can c the prece with bracke

Not Supp

It is not necessary to use set operators with a diff tables.

You can use them with the same table.

```
Select job from emp where Deptno=10
Minus
Select job from emp where deptno=20;
```

o/p

Job

president.

```
Select job from emp where Deptno=10
intersect
Select job from emp where Deptno=20;
```

o/p

Job

Clerk

Manager.

NOTE - Set operators can be used on any no of SELECTs But order by clause will be after the last SELECT

Exn is Top to Bottom.

can change the precedence with brackets.

Not supported by MySQL.

```
Select ....
union
(select ....
minus
select ....
union all
(Select ....
interget
Selectet ....
order by x;
```

upto 255 select

MySQL will always execute multiple SELECTs in top to bottom

DATE

MySQL - SQL INDEXES

- Present in all DBMS and RDBMS.

Emp

ROWID	EMPNO	ENAME	SAL	DEPTNO
X001	5	A	5000	1
X002	4	B	6000	1
X003	1	C	7000	1
X004	2	D	9000	2
X005	3	E	8000	2

Select * from emp Where empno=1;

↓ To Server

Compile

plan

execute.

↓

It will start searching the Emp table. (Always from the start)

even after Empno=1 is found it wont

Stop searching

↓

It will perform full table scan.

If we have millions of rows, it will be way slower.

classmate

PAGE

DATE

∴ WHERE clause is responsible for Searching.

- Indexes are used to Speed up SELECT Statement with a WHERE Clause.

It is Used when we have large no. of ROWS.

- Indexes Speed up Searching.

~ to index of book:

- instead of searching in 1000 pages we search the index. and get the pageno.

- just like page no, in RDBMS. we have ROWID
- we can use this ROWID to create an index.

IND.EMPN — Generally it is a column used for searching

ROWID	EMPNO
x003	1
x004	2
x005	3
x002	4
x001	5

Sorted automatically when index is created.

Index will preserve duplicate values if there are any.

rays from the start)

- W/o index, Searching is performed on main table EMP & that too it is a full table scan. (i.e. it wont stop even when matching row is found)

- In IND.EMPN, Searching will stop when ~~it is found~~ (~~Not a full table scan~~) a non matching row is encountered.

In other RDBMS:

Use index ind-empno;

Select * from emp where empno=1.

(we need to manually invoke index before use.

PAGE

classmate

DATE

In Oracle & MySQL, indexes are automatically invoked by MySQL/Oracle as and when Required

- Duplicate values are stored inside index.

Select * from emp where ename = 'B';

↳ it wont work with earlier index.

∴ Create an index for ename.

Read;
compile
plan. (includes how to execute)
⋮
execute.

known as execution plan.

Execution plan is the plan created by RDBMS as to how it is going to execute your SELECT statement.

↳ plan is like a flowchart.

- In MySQL we can actually see the exec plan
- In Oracle we can change the exec plan.

plan includes checking like:
- does table exist?
- does user have permissions?
- does searching col is present?
- index present?
etc.

classmate

PAGE

DATE

- Oracle Query Optimizer is a product from Oracle Corporation to change execution plan.

* Create an index for column used in WHERE clause.
  ↳ o/p will come in seconds.

- Suppose in future we add new row in table, or update a row, index needs to be updated. (∴ Resorting)

In other RDBMS,
When we
insert/update/delete.....;
~~the~~ we have to write
REINDEX;

In oracle & MySQL,
- indexes are automatically updated by MySQL/Oracle for all your DML operations.
- ∴ MySQL and Oracle are self managing RDBMS.
- ∴ In oracle, you just create the index & forget about it!

DATE

select * from emp where ename = 'B';

(To make it run faster, create an index for ename.

IND. ENAME

ROWID	ENAME
X001	A
X002	B
X003	C
X004	D
X005	E

Select * from emp where sal > 7000;

(To make it run faster.....
index for sal.

IND-SAL

ROWID	SAL
X001	5000
X002	6000
X003	7000
X005	8000
X004.	9000

We have 3 indexes for 0 table (EMP).

- There is no upper limit on the number of indexes per table. (mySQL & Oracle)
- Create index for all cols. that are used in WHERE clause.

classmate

DATE

Ms SQL Server allows only 255 indexes per table

Drawback of indexes:

if you are having large no. of indexes for a table, the slow DML operations will be slowes. (insert, delete) update)

- When you insert in main table (Emp). you need to update all the rows in indexes. will be done automatically

∴ inserts will be slowes.

BUT SELECT WILL BE MUCH FASTER.

In real life,

- SELECT must be faster.
- No body cares if insert, update, & delete is slow
  eg: Booking flight. (We take more time for planning)

- Indexes can be created of any columns with any data type. except TEXT and BLOB.
- You can not index TEXT & BLOB (Huge datatype)

- If you have a null value in a column that is indexed, it is will not be stored in index.
- null values are not stored in an index.

Select * from emp where empno is null;

↳ it will be very slow.

∵ index doesn't store null, RDBMS will search for null in emp. & even search on main table uses full table scan.

PAGE

classmate

DATE

How to make above SELECT faster?

- Instead of storing NULL value in a column, store ~~null~~ a '0'.
- 0 will be stored in index.

If you have two or more independent columns in WHERE clause, create separate index for every column.

```
Select * from emp where empno=2;
```

will use index of empno

```
select ..... sal>5;
```

will use index of sal.

```
Sect * from emp
  where empno=2 and sal >500;
```

These are independent columns.

will use both indexes (empno + sal)

DATE

Composite index

Parent-child colums like deptno, empno.

∴ emp table uses dept no

Combine two or more INTER-DEPENDENT columns in a single index. it is known as composite index.

deptno is parent col.

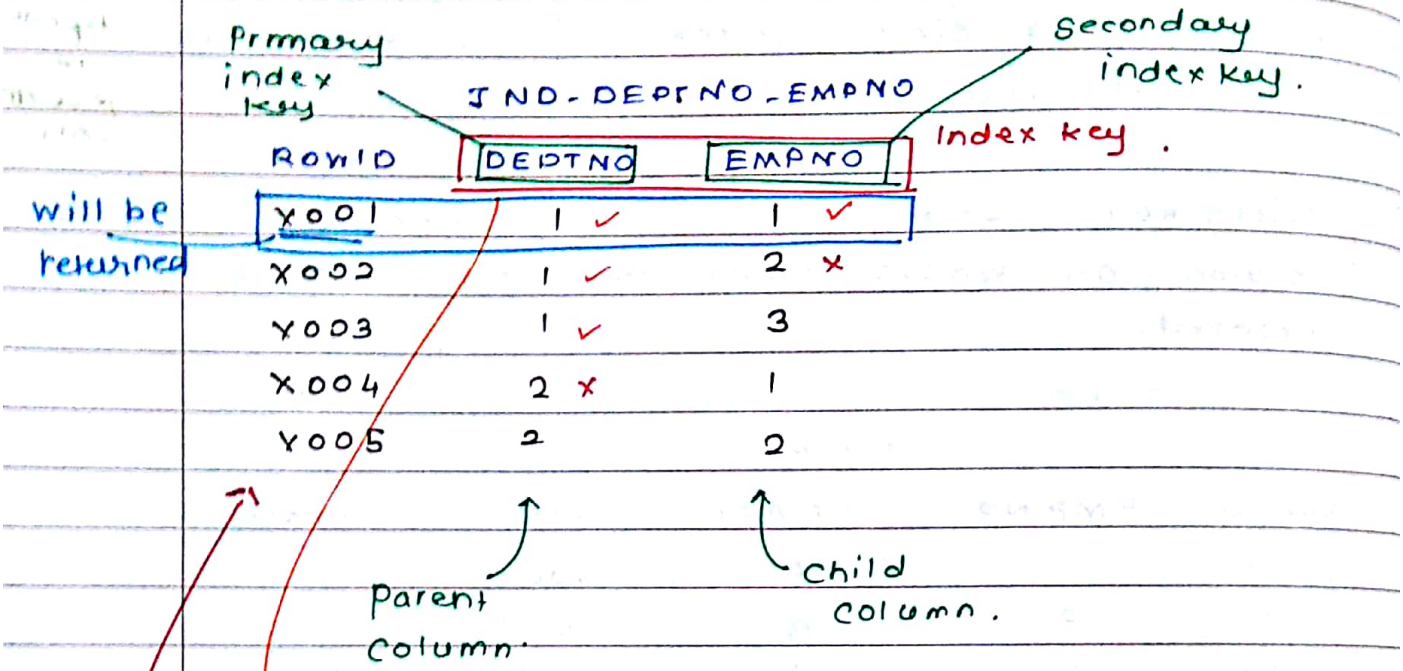
Index key: Index key is a column or a set of column, on whose basis the index has been created.

EMP

ROWID	EMPNO	ENAME	SAL	DEPTNO
X001	1	A	5000	1
X002	2	B	6000	1
X003	3	C	7000	1
X004	1	D	4000	2
X005	2	E	8000	2

- The order of column is important in composite index
- We can combine upto 32 columns in a composite index (MySQL)
- in oracle .... 16 ...

Select * from emp
Where deptno=1 and empno=1;



✗ - Search stops.

Why DEPT. NO is Primary index?
↳ because it is a parent column.

EMPLOYEES are INSIDE DEPARTMENT
DEPARTMENT IS NOT INSIDE EMPLYEES

This INDEX will be invoked only when deptno & empno come together in WHERE clause.
We'll still need to create separate index for DPTNO & EMPNO

DATE

When an index should be created?

- To speed up SELECT statement with a WHERE clause, & ORDER BY clause, distinct, group by, union, intersect, minus.

∵ distinct internally uses sorting.
index is ~~sorted~~ pre-sorted.
∴ Sorting time is saved.

group by also uses sorting internally.....
↳ Not for union all.

```
Select deptno, sum(sal) from emp
    group by deptno;
```

↑ To make this work faster, create index on deptno column.

union, intersect, minus.

↳ ~~They~~ suppress ~~the~~ duplicates.
To suppress dupplicates, they internally use sorting.

PAGE

DATE

If your SELECT returns less than 25% of table data, you should create index.
If it returns more than 25% ....
it would be slower.

SLOW

```
select * from emp where empno > 1;

Slect * from emp where empno=1;  — fast
                       empno=5;  — fast
                       empno<2;  — fast
```

- PRIMARY KEY and UNIQUE COLUMNS should always be indexed. (Roll_no, Ph_no, Pan_no)

DEPT

ROWID	DEPTNO	DNAME	LOC
Y011	1	TRN	Bby
Y012	2	EXP	Dlh
Y013	3	MKT	Cal

EMP.

ROWID	EMPNO	ENAME	SAL.	DEPTNO
X001	1	A		1
X002	2	B		1
X003	3	C		1
X004	4	D		2
X005	5	E		2

classmate

PAGE

DATE

drives. driving

```
Select dname, ename from emp, dept
    Where dept.deptno = emp.deptno;
```

We are trying to make above join faster by having driving table with lesser nos of rows than the drived table.

To make it more faster, make two ~~driving~~ indexed (driving index, driven index).

- Common columns in join operations should always be indexed.

IND_EMP_DEPTNO	
ROWID	DEPTNO
X001	1
X002	1
X003	1
X004	2
X005	2

IND_DEPT_DEPTNO	
ROWID	DEPTNO
Y011	1
Y012	2
Y013	3

(you can create composite index here, these are two columns of different table.

∴ There are two ways of making query run faster:

- having driver with less no. of rows
- by using indexes.

PAGE

Command to create an index.

```
create index indexname on table (columnname);
```

```
create index i_emp_empno on emp(empno);
```

This will create an index for empno
- will do sorting automatically.

Give meaningful name.

I - EMP_ EMPNO

ROWID	EMPNO
X001	1
X002	2
X003	3
X004	4
X005	5

```
create index i_emp_ename on emp(ename);
```

Creating Composite indexes.

```
create index i_emp_deptno_empno on
    emp(deptno, empno);
```

Parent column (eg. country) ⇓ Primary index key

Child column (eg. State) ⇓ Secondary index key

classmate

PAGE

DATE

To drop the index

```
drop index i_emp_empno on emp;
```

Do not do this ↓

```
Select * from i_emp_empno;
```

↗ You can't see the index.

To see all the indexs on table:

```
Show indexes from emp;
```

↳ will show all the indexes on emp.

When we exe. Create table....

↳ table name and otherinfo is inserted to system table.

RDBMS keeps track of everything we define.

To see all the indexes on all tables.

```
use information schema;
Select * from Statistics;
```

↳ Sys. table.

Keeps track of all the indexes.

If you drop a table, then indexes on that table are dropped automatically.

If you drop a column, then indexes on that column are dropped automatically.

PAGE

classmate

DATE

All the indexes are ascending order by default to make it descending:

create index i_emp.empno on emp (empno desc).

ORDERS

ONUM

1

2

3

⋮

100000000

order no.

Most of the time we will work on recent orders.

∴ In cases like `ONUM; create index in desc. order.

Create index. i_orders.onum on orders (onum desc).

DATE

```
create index i_emp_deptno_empno on emp(deptno desc,
                                       empno);
```

while creating an index, we have an option of creating 'unique' index.

```
create unique index i_emp_empno on emp(empno);
```

↳ It works like normal index only.
plus,

WORK OF PRIMARY KEY { it will not allow user to insert duplicate values in empno.

```
insert into emp values(5,'F',5000,2);
```

↳ Will get an error!

error will be:

While creating unique index, what if the column already has duplicate values in the table?

↳ - At the time of creating unique index, if you already have duplicate values in the table then it will not allow you to create the unique index

DATE

- We can have only one index on one column.
i.e. You can not create more than one index on the same column unless, it is combined with some other column (composite column)

- Indexes are of three types:
  1- Normal index
  2- Unique index
  3- Clustered index

Assignment
SQL Exercise 1-6
SQL Asgn 1-22

1. SELECT.... to display the 5th largest SAL (nth.)

can be done be subqueries upto 5 levels.
There is more efficient way. → Find out.

2. SELECT.... to display the largest and second largest sal next to each other.

FIRST	SECOND
.....	.....

3. SELECT.... to display the experience of the employees, in years, remainder months, remainder days.

eg:
3 years 7 months 19 days.

use sysdate() - hiredate.

classmate

PAGE

DATE

- consider leap years.
- O/p should be as per the real life calendars.

HINT: Refer date documentation.

4. SELECT .... To display the ranges of missing EMPNOS.

eg:

EMPNO	O/p Should be.
5	
9	2-4
1	6-10
24	10-16
17	18-23

5.

BANK

DEPOSIT	WITHDRAWAL
5000	3000
6000	500
4500	2000.

SELECT .... To display the balance (cummulative)

O/p Should be.

2000
3000
5500

DATE

6. DELETE .... to delet duplicates rows on basis of empno.

EMP

EMPNO
1 ✓
1 ×
1 ×
2 ✓
3 ✓
3 ×
4 ✓
4 ×
:

HINT: Concept OF ROWID.

Use Subquery.

DATE

MySQL - SQL ALTER TABLE

- ALTER TABLE is a DDL command.

EMP

EMPNO	ENAME	SAL
101	SCOTT	5000
102	KING	6000

- Can rename a table
- CAn add a column to a table
- Drop a column
- Increase width of column.

} can be done directly.

- Can reduce the width of column
- Can change datatype of column
- Copy data from one table into another existing table
- Copy a table
- Copy structure of table
- rename a column
- Change order of columns in table structure.

DATE

Why would you want to change the order of columns?

→ To keep columns with ~~more~~ largest NULL at the end.

Select * from emp; ← Not recommended.

→ O/p order of columns will change when table structure is changed

select ename, sal, empno from emp; ← Recomended

insert into emp values (....); ← Not recommended

insert into emp(ename, sal, empno) values(...); ← Recommend

→ would work the same ~~i.e~~ will provide even after table structure is change.

DATE

rename a table.

- With rename command (DDL command) — will commit automatically

```
rename table emp to employee;
```

renaming a table is not supported by ANSI. (using rename).

- It is not recommended to rename a table.
- It will affect many programs, queries.

Adding a column.

- Why? in future, the new requirement may arrive. (e.g. GST)

```
alter table emp
    add gst float(7,2);
```

o/p.

EMPNO	ENAME	SAL	GST

By default it is added at end. It is full of null values by default. ∴ It won occupy any space.

DATE

deleting/dropping a column:

→ Not done in real life.

```
alter table emp drop column gst;
```

DDL
(∴ Autocommit)
∴ Can't rollback! it becomes permanent.

Increasing width of column.

ENAME varchar(25).

if we want to make it (30),

```
alter table emp modify ename varchar(30);
```

→ will increase width to (30).

```
alter table emp modify ename varchar(20);
```

→ oracle will not allow reducing width.

→ In oracle, we can reduce the width provided that the all the values in column are null;

→ MySQL will allow this directly, but the data will get Truncated (Not safe)

DATE

update emp set ename = null;
alter table emp modify ename varchar(20);

Will make all the ename values null first. & then alter table will reduce the width.

But all ename data will be lost.

∴ Before making all ename null, make a back up of ename column.

alter table emp add X varchar(25); — backup column 'X'

Backing up (copying)
update emp set X = ename; ~~ename = null~~
updat emp set ename = null;
alter table emp modify ename varchar(20); — reducing width.

/* data testing
testing if the data is
not exceeding 20
*/

update emp set ename = X; — restoring.
alter table emp drop column X; — deleting

It is a good idea to not to delete 'X'.
For future manipulations.

- While creating the table add extra empty columns for future.

PAGE

DATE

- While creating a table, add extra columns in advanced. These columns wont consume space (∵ they are Null).
- They can be used in future manipulation.

- It is a good idea to have few extra columns in table
  → They are called Extension Columns. (used to extend the table).

- Just like having extension column, we can have extension table. (used to extend the application). → For future.

  e.g. for EMP we can have X-EMP as an extension table.

Chan

DATE

Changing datatype of column.

EMPNO char(4).

We want to make it int(4)

Changing data types is not recommended.

alter table emp modify ename varchar(20);

alter table emp modify empno int(4).

To be safe.

alter table emp add x char(4);
update emp set x = empno
update emp set empno = null;
alter table emp modify empno int(4)

/* data texting on x column.
converting char data to int etc */

alter table emp drop column x;

In Oracle, we can change the data type of column provided the contents are null.

DATE

copying data from one table into another existing table

EMP

EMPNO	ENAME	SAL
101	SCOTT	5000
102	KING	6000

EMP2

EMPNO	ENAME	SAL
3	QUEEN	4000
4	JACK	3600
5	ACE	5000

Identical structure.

We want to copy data from EMP2 to EMP1

```
insert into emp select * from emp2;
```

o/p

EMP

EMPNO	ENAME	SAL
101	SCOTT	5000
102	KING	6000
3	QUEEN	4000
4	JACK	3000
5	ACE	5000

Sub query with insert

Do not use 'VALUES' and brackets.

This select wont show any o/p.

DATE

copy a table.

We want to copy data from EMP table to a new table with some structure as EMP
EMP_COPY table

Why to have a copy of table?
for back up?
↳ NO!
For backing up table, we have EXP (Export) command.
↳ will create a dump file (.dmp)

So, why?
- For Testing. (S/W testing).
- Testing is always done on a copy. It is not done on original data.

Subquery with create table.

```
Create table emp_copy
as
Select * from emp;
```
↳ o/p will not come in screen.

i) emp_copy is created based on select stmt.
(i.e structure of emp_copy = struct. of emp)

ii) SELECT is executed; o/p of SELECT will not be displayed on the screen; o/p of SELECT is inserted into emp_copy.

PAGE

DATE

copy the structure of table (Not rows)

↳ There are 3 methods to do this.

i)

```
create table emp-copy
as
select * from emp;
```

↳ creating copy of emp. (with rows)

```
delete from emp-copy;
```

↳ deleting rows (so we are just left with structure)

not auto commit

```
commit;
```

∴

Truncate

↳ It is a DDL command: (auto commit).

- It is similar to DELETE.
- It will delete all the rows but it wont delete the table
- ∴ it will preserve the structure of table after deleting all the rows.

DELETE	TRUNCATE
- Delete requires commit	It is autocommit
- DML command	DDL command
- We can use WHERE clause	WHERE clause is not allowed.

DATE

ii)
```
create table emp-copy
as
select * from emp;
```
creating copy

```
truncate table emp-copy;
```
Truncating rows.

iii) By using impossible WHERE clause.

```
Create table emp-copy
as
select * from emp Where empno = -1;
```
SELECT will return 0 rows.

~~But, if the~~
To make WHERE clause even more impossible,

```
create table emp-copy
as
Select * from emp. Where ~~emp~~ 1 = 2;
```

DATE

rename a column.

→ Does n't happen in real lif.

```
Create table emp_copy
as
Select empno, ename, sal as Salary
 from emp;

drop table emp;

rename table emp_copy to emp;
```

changing name.

EMP_COPY → emp

EMPNO	ENAME	SALARY
101	SCOTT	5000
102	KING	6000

DATE

changing order of columns in table structure

↳ because of null values

```
create table emp-copy
as
select ename, sal, empno from sal;

drop table emp;

rename table emp_copy to emp;
```

Changing the order.

EMP_COPY EMP

ENAME	SA	EMPNO
:	:	:
:	:	:

NOTE

```
create table emp_copy
as
select ename, sal, empno from sal;
```

When you copy a table,
When you create a table using sub-query, then indexes and constraints of original table are not copied into new table.

MySQL - SQL Constraints

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

* Constraints are limitations / restrictions imposed on a table.

PRIMARY KEY CONSTRAINT.

- KEY is another name for column
- ∴ Primary key -> primary column.
- Primary key is a column or set of columns that uniquely identifies a row.
- In above e.g. (EMP) we can consider EMPNO as PK
- Duplicate values are not allowed. It has to be unique.
- i.e we cant have two employees with the same empno.
- Also, the null values are not allowed.
- User cant give null value to EMPNO.

∴ PK is a mandatory column.

∴ user has to enter PK. value.

- Puspose of PK is uniqueness.
- With EMPNO we cant distinguish bet' to employees.

DATE

- Having PK is not compulsory in table.
- It is recommended that, every table should have primary key.
- It helps in a long run.
- MySQL internally distinguishes two rows with 'rowid'.
- PK is like user's ROWID. (user defined).
- We can have missing values in PK.
- PK can be alphanumeric.

So, the purpose of primary key is ROW UNIQUENES.

- With the help of primary key, we can distinguish between two rows.

In case of

EMP

ENAME	SAL
...	...

Which column to be made PK.

We cant make ENAME & SAL as PK.

∴ add a column

EMP

COL-1	ENAME	SAL

If you can not identify a column for PK.

add a new col.

Can we make have ROWID as a Primary key?

- NO.
- ROWID is not a column of a table.
- It is a pseudo column.

DATE

Why not add rowid in col-1 and make it PK?

↳ - ROWID is a string in encrypted form.
- Even if we make it PK, we want be able to understand the significance of it.
- And ROWID is not constant. it may change when a row is updated.

- Any DT column can be PRIMARY KEY.
- BUT,
- TEXT and BLOB can not be PRIMARY KEY.
-

IF EMPNO is PK,

```
select * from emp where empno = 1;
```

it is guaranteed that this SELECT will return only 1 row.

∴ PK column is best column for searching.
- In MySQL & Oracle, Index is created automatically for PK column.
- YOU CANNOT DROP this automatically created index.

This is called Unique index

DATE

* Lets say EMPNO is not PK.

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	A	6	1
3	C	7	1
1	D	9	2
2	E	0	2

Composite PRIMARY KEY :) Combine 2 or more inter-dependent columns, together to serve the purpose of PRIMARY KEY.

In EMP, ∵ EMPNO has duplicates we cannot make it PK.
But combination of EMPNO, DEPTNO (they are inter-dependent) can ensure uniquenes.

EMPNO + DEPTNO → composite PRIMARY KEY.

In Oracle → can combine upto 16 columns in a composite PRIMARY KEY.

In MySQL → .... 32 Columns

∴ Before going for Adding a New column as PK. Check if your table has any inter-dependent columns.

PAGE

DATE

- If you declare a composite PRIMARY KEY, in the index that is created automatically would be composite unique index.

Steps for identifying primary key:

i) Examine the structure of table, identify some key columns, and make it the PRIMARY KEY of your table.

ii) If you cannot identify key column, try for composite PRIMARY KEY: (try to restrict Composite PRIMARY KEY to max 4 or 8 columns otherwise it becomes inefficient / very slow).

iii) If you cannot identify composite PRIMARY KEY, add extra column to the table to serve the purpose of primary

SURROGATE KEY -> is PRIMARY KEY.

SURROGATE KEY -> if you can not identify PRIMARY KEY in the table, add an extra column to the table and make it the PRIMARY KEY of that table. Such a PRIMARY KEY (column that has been derived from outside the table) is known as SURROGATE KEY.

- For SURROGATE KEY, char datatype is recommended (because its a good column for Searching)

DATE

PK

EMP

EMPNO	ENAME	SAL	DEPTNO	PAN_NO	PH_NO
1	A	5000	1	:	:
2	A	6000	1	:	:
3	C	7000	1		
4	D	9000	2		
5	E	8000	2		

- We can have only one PRIMARY KEY CONSTRAINT per table.

- PAN_NO & PH_NO are also candidate for PK.
- In future, suppose we drop EMPNO. The PAN_NO, or PH_NO can become PRIMARY KEY.
- ∴ It is good to have more than one candidates for pk.

- PAN_NO & PH_NO are CANDIDATE KEYS. (in future they can become PK).

- CANDIDATE KEY is not a constraint
- CANDIDATE KEY is a definition
- CANDIDATE KEY → besides the PRIMARY KEY, any other column or set of columns that can also serve the purpose of PRIMARY KEY, is a good candidate for PRIMARY KEY. is known as CANDIDAE KEY.

CANDIDATE KEY → Future primary key in waiting.

Command to create PRIMARY KEY.

. We can define PK at the time of table creation.

```
create table emp
(
    empno  char(4) primary key,
    ename  varchar(25),
    sal  float(7,2),
    deptno int(2)
);
```

→ consider EMP table from prev page.

```
insert into emp values('5','F',5000,2);
```

MySQL/ Oracle/ RDBMS will not allow this insert. (Violation of PK)

```
insert into emp values(null,'F',5000,2);
```

→ Wont allow.

- All the constraints at server level.
  it means, you may perform any DML operation (insert, update, delete) using any front-end S/w (java, html, .net ...) the constraints will always be valid.

DATE

What is a constraint internally ?

→ internally constraint is a system defined function which performs validation. (checking for duplicates, nulls etc...)

To find out which all constraints you have are there on table ?

→ info about constraints are stored in System table.

Select * from information_schema.table_constraints

This is a sys table.

will show all constraints, on all tables, on all schema.

Select * from information_schema.table_constraints
Where table_schema = 'DAC1';

(your schema name.)

will show the PK constraint but wont tell on which column.

(all const.)

Select * from information_schema.key_column_usage
Where table_name = 'EMP';

DATE

If we make empno as PK, index will be created for empno. we can see it using

```
Select indexes from emp;
```

To drop PK; (Data will remain. only constraint is dropped)

```
alter table emp drop primary key;
```

- What if we have an existing table EMP & want to make empno PK?
  We can add PK afterwards by:

```
alter table emp add primary key (empno);
```

Before adding PK, what if empno has duplicates / nulls ?

mySQL won't allow this!

* When the PRIMARY KEY (or unique constraint) is dropped, then the ~~INDEX~~ associated System created index is dropped automatically.
Later if you add the primary key constraint or unique constraint then index for that column is created automatically.

DATE

How to create composite PK.

```
create table emp
(
    empno char(4),
    ename var char(25),
    sal float (7,2),
    dept no int(2)
    primary key (dept no, empno)
);
```

Composite primary key has to be specified at the end of the table structure

The order of columns is important.

It will affect the index (composite)

- It will affect the speed of searching
- Ideally, parent column should come first & the the child column.

Constraints are of 2 types:

a) Column level constraint : constraints specified on single column

b) Table level constraint: Constraint specified on 2 or more columns. it is a composite constraint. & it has to be specified at the end of the table structure.

NOT NULL constraint

- Just as name suggest, Similar to primary key NULL values are not allowed

  BUT

  Duplicate values are allowed.

- You can have any no. of NOT NULL constraints on a table unlike PK.

- There is no such thing like composite NOT NULL
- It is always a column level constraints. We cannot have composite not null constraints.

```
Create table emp
(
  empno char(4),
  ename varchar(25) not null,
  sal float(7,2) not null,
  dept no int(2)
);
```

- In mySQL, nullability is a part of the data type.

- To see not null constraint columns :-
  - Sys. table doesn't store info about not null const.
- You have to describe (desc) the table to see it

DATE

To drop not null constraint.

```
alter table emp modify ename varchar(25)
                                    null;
```

To add not null constraint.

```
alter table emp modify ename varchar(25)
                                    not null;
```

We can not create more than one primary keys.

- Indirectly we can have more than 1 primary key per table.
- 1 primary key can be created directly (as seen before.) eg(empno).
- and ~~remaining~~ for remaining columns, specify not null constraints + unique index.

will indirectly make PK.

do this to PAN_NO, PH_NO.

They will become indirect PKs.

DATE

-> ~ to PK

-> diff than PK

UNIQUE constraint.

- ~ to primary key in several aspects.

-> - It will not allow duplicate values. Similar to primary key.

- BUT

-> It will allow null values.

e.g: MOBILE NO, BANK ACC NO.

- You can insert any no. of null values.
  (-> Special treatment given to null values.

EMP

EMPNO	ENAME	SAL	DEPTNO	MOB_NO

-> - Text and BLOB can not be UNIQE.

-> - UNIQUE index is automatically created

-> - UNIQUE column is good for searching.

-> - we can have composite UNIQUE (Empno + Deptno)

-> - Upper limit for combination of columns:
  Oracle -> 16 columns
  MySQL -> 32 columns.

-> Unlike primary key, we can have any no. of UNIQUE constraints per table..

DATE

```
create table emp
(
    empno  char(4),
    ename  varchar(25),
    sal    float(7,2),
    dept_no int(2),
    mob_no char(15) unique,          ← column level constraint
    unique(dept_no, empno)           ← Table level constraint
);
```

composite constraint at end.

```
Select * from information_schema.key_column_usage
Where table_name = 'emp';
```

- Unique index automatically created for unique col.
  - (1) show indexes from emp;
    index name will be same as col. name.
  - -> Composite index takes first column name (deptno) as the name of composite index.

- Unique constraint is also an index so to drop it use :-

```
drop index dept_no on emp;
drop index mob_no on emp;
```

NOTE :

```
Create ...
(
  :
  :
mob_no char(15),
unique(dept_no, empno),
unique(mob_no)
);
```

Preffered approach: ↑readable
Writting constraint at end.

* A column level constraint can be written at table level but a table level constraint cannot be written at column level.
* column level constraint can be written at table level except for not null constraint which is always a column level & hence syntax will not support writting not null constraint at table level.

PAGE

DATE

To add unique constraints to existing table.

```
alter table emp add constraint u_emp_mobno
    unique (mob_no);
```

'constraint u_emp_mobno' -> optional.

if you dont specify constraint u_emp_mobno then MySQL will give it's own name to the constraint (which happens to be the same as that of the column name due to which created)

Alternate soln for ~~PRIMARY~~ CANDIDATE KEY.

PAN_NO -> notnull constraint + unique constraint

- One column for can have multiple constraints.

it becomes ∿ to PRIMARY KEY.

For CANDIDATE KEY column, if you specify 'not null' constraint + 'unique constraint, it works similar to PRIMARY KEY.

- Candidate key column has now become an alternative to PRIMARY KEY.
- Such a candidate key column is known as ALTERNATE KEY.

ALTERNATE KEY is not a constraint, it is a def^n.

Lets say Empno -> PK -> direct PK.

Passport_No -> ALT. KEY. } indirectly.

PAN-NO -> ALT key.

Logically we have 3 PKs.

If you have alternate keys in your table, then primary key (column) is a ~~SUPER KEY~~ known as SUPER KEY.

SUPER KEY is not a constraint, it is a definition.

NOTE: To specify two or more constraints to a column at the time of table creation,

```
Create ...
...
    passport-NO  char (15) not null unique,
... ;
```

DATE

EMP

EMPNO (Parent col, Key)	ENAME	SAL	DEPTNO (FK, child col)	MGR (FK, Child column)
1	A	5000	1	1
2	A	6000	1	1
3	C	7000	1	1
4	D	9000	2	2
5	E	8000	2	2
6	F	6000	2	2

Foreign key constraint / column

- key means column.
- ∴ it is a foreign column.

DEPT (PK / UNIQUE, parent col)

DEPTNO	DNAME	LOC
1	TRN	Bby
2	Exp	Dlh
3	MKT	Cal

- DEPTNO column is common in both the tables
- On basis of this comm. col^{n}, we wrote joins.

Our requirement is:

- User can only insert row with DEPTNO as 1 or 2 or 3.
- it should not allow insert like

7 G 7000 99 1 2

Depth

DATE

Foreign Key is a column or set of columns that references a column or set of columns of Some table.

deptno of emp references dept no of DEPT. → is dependent on

∴ dept no of DEPT is parent column.
& " of Emp is child column.

Foreign Key constraint is specified on the child column. (i.e. the dependent column) not the parent column.

- Foreign Key constraint is not used for joins!

- Parent column has to be PRIMARY KEY or UNIQUE (can have null.) (But no duplicate values).

→ eg:

Dept NO	DNAME	...
1	TRN	
2	Exp	
2	MKT	

Ambiguity

∴ it has to be PRIMARY Key or UNIQUE.

Comparison with null will return null
∴ it is OK to have nulls in parent column

This is pre-requisite for FOREIGN KEY constraint.

PAGE

DATE

Deptno of Emp (FK column) has duplicates.
i.e:
- Foreign key column (child column) will allow duplicate values unless specified otherwise.

- Foreign key column (child column) will allow NULL values unless specified otherwise.

- EMPNO and MGR column in EMP are inter-dependent
- EMPNO is parent col. & EMP MGR is child column

- Foreign key column (child column) can reference a column of same table also. This is known as SELF-referencing.

→ Redd the defn of FK now, with underlined part!

Commands for FK

```
/* creating parent table */

create table dept
(
 deptno int (2) primary key,
 dname varchar (15);
 loc varchar (10);
);
```

commands for FK:

/* creating child ~~parent~~ table */

```
create table emp
(
    empno  char(4) primary key,
    ename  varchar(29),
    sal  float(7,2),
    deptno  int(2),
    mgr  char(4),
constraint fk_emp_deptno foreign key(deptno)
        references  dept(deptno),

constraint fk_emp_mgr foreign key(mgr)
        references  emp(empno)
);
```

constraint name (optional)

constraint fk_emp_deptno
constraint fk_emp_mgr
} optional. but makes it more readable.

Now, if you try to insert

7 9 7000 99 2

7 G 7000 2 2S

Will not allow due to foreign key constraint

DATE

```
Select * from information_schema.Key_column_usage
    Where table_name = 'EMP';
```

To drop FK constraint

```
alter table emp drop foreign key fk_emp_deptno;
```

name of constraint.

To add foreign key in future:

```
alter table emp add constraint fk_emp_deptno
    foreign key (deptno)
        references dept (dept no);
```

Optional.

You can not delete parent row (Master row) when child row (details row) exists

eg: delete from dept where dept no=2;

Trying to delete a department which still having employees in it.

DATE

If you want to delete a parent row & on deleting parent row, you also want to delete its child rows, then:

```
create table emp
(
  :
  :
constraint fk_emp_deptno foreign key (deptno)
references dept (deptno) ON DELETE CASCADE,
  :
  :
);
```

∴ delete from dept where deptno = 2;

↳ Will now delete 2nd row from dept table & all the employees in department 2 from emp table.

ON DELETE CASCADE: if you delete the parent row, it will automatically delete child rows also.

To preserve child rows after deleting the parent row. by making them null.

update emp set deptno = null where deptno = 2;
delete from dept where deptno = 2;

DATE

* - You cannot UPDATE parent column, if child rows exist.

```
Update dept set deptno = 4
                Where deptno=2;
```

→ It wont allow. this update.

* ON UPDATE CASCADE -

IF you update the parent column then it will automatically update the child rows also.

```
Create table emp
(
   :
   :
constraint fk_emp_deptno foreign key (deptno)
references dept(deptno) ~~ON DELETE CASCADE~~
                        ON UPDATE CASCADE,
   :
   :
);
```

```
Update dept set deptno=4
                Where deptno=2;
```

→ This is allowed now.

Now, the deptno of dept is updated to 4 & All the child rows in emp (rows with deptno 2) are also updated to 4.

DATE

ON DELETE CASCADE

PK

parent

EMPNO	ENAME	SAL	DEPTNO	MGR
1				
2				1
3				1
4				2
5				2
6				2

FK

child

What will happen if first row is deleted?

- All the rows will be deleted.

ON DELETE CASCADE is not recommended in the event of self referencing. You may end up deleting more rows than expected.

CHECK constraint

- It is used for validations. (Used for checking purposes)
- e.g. sal > 5000 (user has to enter sal > 5000)

```
Create table emp        IGNORE THIS
(
   empno  char(4),
   ename  varchar (25),
   Sal  float (7,2),
   dept no  int(2),
   Status  char(1),
   comm  float (7,2),
   mob_no  char (15)
);
```

table w/o any validations.

Ideally, every column should have a constraint. It will give a very powerful structure.

column level constraint.

```
create  table  emp
(
  empno  char (4) auto_increment  PRIMARY  KEY
  ename  varchar (25), check (ename = upper (ename))
  Sal  float (7,2), default  7000  check (sal > 800 and sal < 9000)
  dept no int ( 2), references  dept(deptno),
  Status  char (1), default  'T'  check (status in ('T', 'P', 'R'))
  comm  float (7,2), not null,
  mob_no  char (15) unique,
  check (sal + com < 180000)
);
```

→ table level constraint.

DATE

To make use of auto-increment and default values, use the following syntax for INSERT statement:-

insert into emp (ename, deptno, comm, mob-no)
value (....);

↑ We are not specifying inserting empno, manually ∵ autoincrement.
sal, status } default

default 7000 check (sal 75000 and sal < 90000)

- ↳ If user does not enter any value, the default value inserted will be 7000.
- And, if user inserts a value, it has to be between 5000 and 90000, else sys. will give error!

* With 'check' we can use:
  - Relational operators
  - Logical operators
  - Arithmetic operators
  - Special operators like (IN, BETWEEN, LIKE etc)
  - All single row functions like (upper ())
    ↳ Not group function!

CHECK Constraint - command works in MySQL (won't give an error on writing
check (sal>5k) on a column.
(Sal 75000).

But the constraint does not work.
will allow SAL < 5000

D:76 L:8

DATE 30/10/2017

CHECK constraint is not enable in MySQL

CHECK constraint works in Oracle.

CHECK constraint -> enabled in Oracle.

When you install Oracle 3 users automatically Created.

i) Scott (pwd : tiger) ⇒ - Regular user (demo user).
- has basic permisions like create table, create procedure, create trigger, ..., etc.
(- has: connect & resource privileges)

ii) System (pwd: manager) ⇒ - Has DBA privileges.. create users, assign privileges, take backups, performance tuning, performance planning, performance management.

iii) Sys (pwd: change_on_install) ⇒ - Super user
- It has DBA privileges & this user is owner of database, system table. (Sys user can drop system table), there are more than 2000 System tables in Oracle. (63 in mySQL.)
- Job of sys user is to
  - Startup the database
  - Shut down database &
  - perform the recovery

DATE

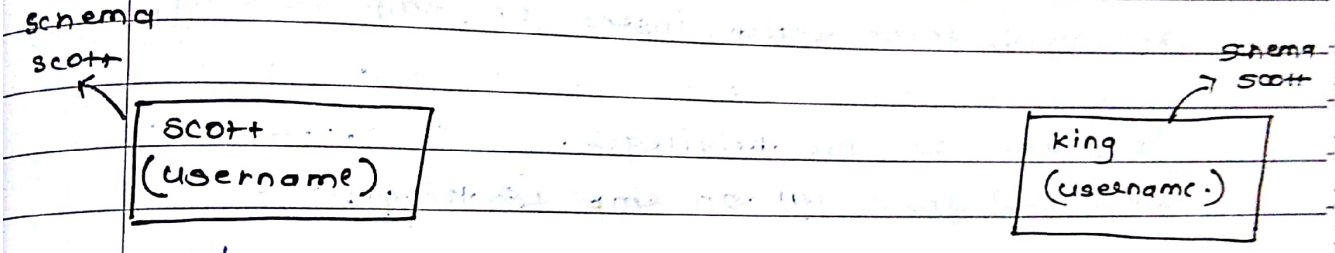
When you install MySQL, only one user is automatically created.

root
(pwd: specified at the time of install) ⇒ Has DBA permissions/privileges.

MySQL - SQL Privileges. (GRANT / REVOKE)

- GRANT / REVOKE are DCL commands.

consider a multi user environment.



↓

EMP
(created by scott)

i.e scott is the owner of EMP

scott can decide whether to allow king to access from EMP.

in command line

SCOTT - MySQL > grant select on emp to king;

↑ means command is executed by SCOTT.

→ Now king can only SELECT from EMP.

You can grant permission(s) to a user(s) to only one table at a time.

DATE

SCOTT_MySQL> grant insert on emp to king;

↳ Now king can SELECT & INSERT to EMP. (But he cant UPDATE)

SCOTT_MySQL> grant updat on emp to king;

SCOTT_MySQL> grant ~~dpdate~~ delete on emp to king;

↳ Now king can SELECT & INSERT UPDATE & DELETE From emp.

To grant mutiple privileges.

SCOTT_MySQL> grant select, insert on emp to king;

To grant all the privileges.

SCOTT_MySQL> grant all on emp to king;

To grant permission (one) to mutiple users.

SCOTT_MySQL> grant select on emp to king, queen, jack;

To grant mutiple permissions to mutiple users.

SCOTT_MySQL> grant select, insert on emp to king, queen, jack;

SCOTT_MySQL> grant all on emp to king, queen, jack;

When you grant permission (permissions) to uses (users), it is permanent (Not for a session).

DATE

REVOKE (opp of GRANT).

To revoke permission from a user.

SCOTT-MySQL> revoke select on emp from king;

To grant permission to all users.

SCOTT-MySQL> grant select on emp to public;

SCOTT-MySQL> grant select on emp to king with grant option;

→ Now, king user can grant/revoke permissions on SCOTT's emp.

KING-MySQL> grant select on scott.emp to queen;

scott → not username, it is Schema name

emp → emp of SCOTT.

QUEEN-MySQL> select * from scott.emp;

Schemaname.tablename.

USERNAME & SCHEMA name need not be same.

But it is recommended for ease of access.

DATE

To see permissions received :-

schema name — table name

```
select * from information_schema.table_privileges;
```

→ information about Permissions granted/Revoked are stored in a system table.

System Tables

- 63 system tables in MySQL.
- In oracle, there are more than 2000 system tables.
- They store complete info about DB. (e.g. users, permissions, constraints, ... etc).

- All system tables are stored in information_schema.

- eg. - Statistics -> table that stores info about indexes.

  - table_privileges.
  - table_constraints
  - key_column_usage

- All the system tables are READ-ONLY. (you can only SELECT)

DATE

* · DDL for user is DML for system tables

→ When we

DDL
create table emp
(
:
);

DML
info of emp is insested in system table

DDL
drop table emp;

info of emp is ~~delered~~ dropped from sys·table.
DML

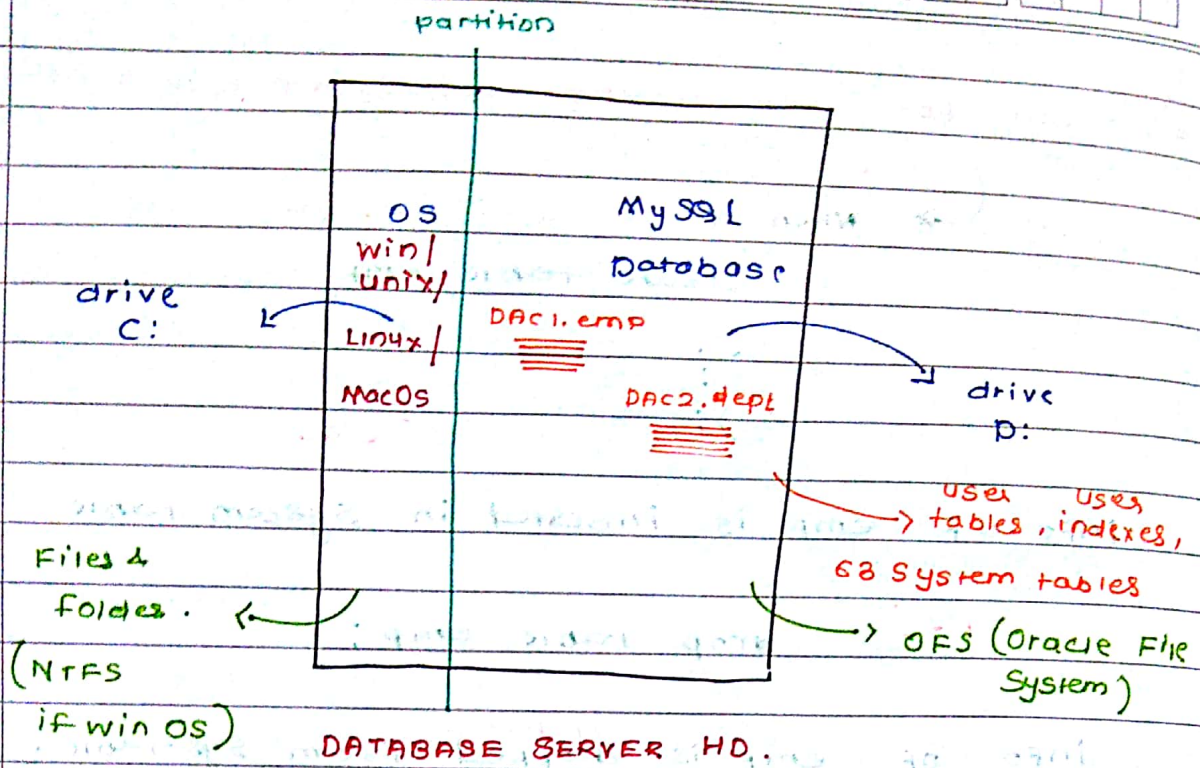
∴ Data is of 2 types:

User data: User created data
Uses tables and indexes.

System data: System created data
data that is stored in System tables.

→ is also known as Metadata.
- Metadata is data about data.

DATE



This is DATABASE Architecture.

- When you install MySQL, DB will be Stored in C: drive ! (in case of default setup.)

- with custom install, create DB on separate drive.
- it is a good idea to keep files, OS & DB on separate drives.

- with custom install, we have an option of formatting a drive to OFS.
  - OFS is based on LINUX kernel.
  - OFS can not be accessed by OS.
  - just like Redhat Linux, Oracle has its own version of Linux.
    - Oracle Linux
      ↳ It is the most secured Linux (unbreakable Linux)

DATE

- Oracle Linux is designed specifically for Oracle and mySQL database.
- Design specifically for security.

MySQL - STORED OBJECTS.

- Objects that are stored in the database
- eg. tables, indexes, view.

Views

- Views are stored objects.
- Concept of views in all RDBMS and some DBMS (MS Excel).

DB

User -> SCOTT Schema -> SCOTT

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

→ SCOTT is the owner of this table.

- A VIEW is a handle to a TABLE.
- It is a hard disk pointer. (Not like pointers of C. C-pointers are in RAM).

→ It stores the add. of table. It is known as a LOCATOR

PAGE

DATE

Why do we create view?

- VIEW is used for indirect access to a table.
- upto now, we have accessed tables directly
  e.g. (select * from scott.emp)

- Why to access table indirectly?
  ↳ For security purposes.

- VIEW is used for security purposes.
- It is used to restrict acess of users.

Lets assume scott user wants to share ~~ename~~ emp to with KING. But scott user wants to share only selective data (e.g only ENAME, empno) this is not possible with GRANT

We can only have SELECT in view.

name of view

SCOTT_MYSQL> create view viewname .... ;

SCOTT_MYSQL> create view V1
as
select empno, enome from emp;

scott will get to see a msg
'view created'

↳ view is stored in DB HD.
it is a stored object.

classmate

PAGE

Table is one, Views are many!
Mumbai's data can be seen by mumbai's users.

DATE

Table

- There is a system table for views also.
- in the sys. table for view, an entry is made.
  when we create a VIEW.

☆ `V1 = SELECT EMPNO, ENAME from emp`

☆ Schema is the synonym for Database.

SCOTT-MySQL > grant select on v1 to king;

↳ Granting the SELECT permission on view to king.

KING-MySQL > Select * from scott.v1;

↓

Schemaname.viewname.

internally ★ SELECT is executed

o/p.

EMPNO	ENAME
1	A
2	B
3	C
4	D
5	E

KING-MySQL > Select * from scott.emp;

↳ ERROR!

↳ King has no SELECT permission on 'EMP'. It has SELECT permission only on view V1.

⇓

This is security!

PAGE

DATE

& encapsulation.

∴ View is used for security. (~ Abstraction)

- Used to restrict column access.
- It is a form of encapsulation.
  (~ to getter())

- Giving user permission on views not on table. } giving access indirectly.

- Diff. user has diff. requirements.
- ∴ create Separate VIEWS for separate users.
- There's no upper limit on no. of VIEWS.
- creating VIEW ≠> creating duplicate table

(↳ View does not contain data!
View contains only the definition.
(V1 = SELECT EMPNO, ENAME FROM EMP)

- once you create a view it is permanent!
- You have an option of dropping views.
- You have an option of viewing all the views you have defined. (From System table)

- You can not UPDATE a view,
- You have to drop a view & create a new view with changes.

- DONT GRANT ANYTHING ON TABLE
- give GRANTS on VIEWS

DATE

if you rename the column in table which is present in view, the view will not work. You'll need to update the view as well. & for updating view, you'll need to drop the existing view & create a new view.

Try to create view for every combination of SELECT

Does w view slow down the performance?

↳ A view will speed up the SELECT!

- with ordinary SELECT

↳ READ
compile
plan
execute.

} Not req^d for views.

- View is stored in DB in compiled format.

∴ the plan is reddy!

★ ∴ View is an executable format of SELECT Statement

∴ Execution will be very fast.

- If indexes are present, it will run even faster!

What is the advantage of having it in compiled format?

↳ Hiding source code from the users.

DATE

SCOTT - MySQL> grant select, insert on v1 to king;

Granting insert to king on v1. ⇒ It is Allowed

KING - MySQL> insert into scott.v1 values(6,'F');

→ v1 (view) does not contain table(data) it just contains a pointer to the table (emp.).

it will pass on values (6,'F') to emp.

- DML operation can be done on a view.
- ∴ DML operations done on view will affect the base table.

- Try to

∴ Create 3 views for insert, update, delete also!

(Try to do everything with VIEWS in your project)

With above insert we can only insert EMPNO & ENAME to emp through v1.

→ Rest of the columns for this row in emp will have null value.

→ What if sal has 'not null' constraint)

→ In this case, View will not work.

PAGE

- To make it work, define sal with 'default' constraint as well.

∴ Constraints that are specified on the table (emp) will always be enforced, even if you insert via view (vj).

- View can only be dropped by its creator (SCOTT in our case.)

You can share the view with many users.

SCOTT_MySQL> grant select, insert on v1 to king, queen, jack

- You cannot 'alter' through view. You have to alter the table directly.
- only owner (SCOTT in our case) can alter the table.

To change the SELECT statement of view:-

SCOTT_MySQL> drop view v1;
SCOTT_mySQL> Create view vj as select.... ;

OR

SCOTT_MySQL> Create or replace view v1 as select.... ;

if v1 is present it will over write it with new select. if v1 is not present it will create v1.

SCOTT_MySQL> create or replace view v2
as
select * from emp where deptno=1;

Creating a new view V2.

V2 = SELECT * from emp where deptno=1

We have a Where clause.

SCOTT_MySQL> grant select on v2 on dac1, dac2;

DAC1_MySQL> select * from scott.v2;

schemaname.viewname

o/p

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1

- User can only see rows with deptno=1.
- So we are restricting the rows now.
- Earlier we were restricting the column.
- ∴ Views are used to restrict the rows.
- User can see only the subset of rows.

DATE

SCOTT.MySQL> grant insert on v2 on dac1, dac2;

↳ granting insert permission on v2 for dac1 & dac2.

DAC1.MySQL> insert into scott.v2 values(6, 'F', 6000, 2);

e.g.
dac1 (CDAC Kharghar) can take admissions on behalf of dac2 (CDAC Juhu).

Show it allow? -> NO.
will it allow? - YES!

To make it work ↓

SCOTT_MySQL> create or replace view v2 as
select * from emp *
Where deptno=1 WITH CHECKOPTION;

DAC1.mySQL> insert into scott.v2 values(6, 'F', 6000, 2);

↳ ERROR!

'WITH CHECK OPTION' is more powerful than 'CHECK'
When we use 'CHECK' on SAL while creating table
... sal float(7,2) check (sal<5000)
With this, All the users are restricted to insert sal <5000.

Amdocs : interview:

∴ view with check option can be used to enforce different checks for different users (Similar to CHECK constraint but more powerful than CHECK constraint).

```
SCOTT_MySQL>Create view v1
            as
            select empno, ename, sal, deptno, gst from emp;
```

But we dont have gst column in emp.
∴ ERROR!

However, if you still want to create a view,

```
SCOTT_MySQL> Create FORCE view v1
             as
             select empno, ename, sal, deptno, gst
             from emp;
```

→ It will force view to be created even if 'gst' doesn't exist.

→ View will be created even if EMP doesn't exist.

→ will give warning "view created with compilation errors"

The SELECT stmt of this view is stored in the sys. table in UNCOMPILED Format.

DATE

After this view is created, if we do:

```
Select * from v1;
```

↳ It will give an error!

Then what is the purpose of such view?

↳ - We can create a view first.
- We can add 'gst' column in future.
- Afterwards when gst column is added, the view will work.

```
/* adding gst column */

alter table emp add gst float(7,2);
```

```
Selet * from v1;
```

↳ It will first compile the Uncompiled view v1 & it will be stored in System table in compiled format.

- It will make the plan
- It will execute.

↳ This will not be done for all next SELECT's

The option of FORCE VIEW is not supported by MySQL.
It is supported by Oracle RDBMS.

PAGE

DATE

```
create or replace view v1
as
select ename, sal from emp;
```

```
desc v1;
```

→ You can describe a view!

View has no data in it but it has a structure.

```
select *. from v1;
select ename from v1;
select * from v1 order by ename;
```

We can also perform joins with views.

```
create or replace view v1
as
select ename, sal*12 as 'ANUAL'
from emp;
```

→ view based on computed column.

```
select * from v1;
```

You cannot insert, update view based on computed columns.

↓

if we try to insert annual into view, it would not divide it with 12 & add it as sal in EMP table

↑ It is supported by Magic
Forte
Smalltalk.

classmate

PAGE

DATE

magic
forte
Smalltalk } OODBMS

MySQL → RDBMS

Oracle → RDBMS + OODBMS
→ ORDBMS (object Relational DBMS)

- From a 'view based on computed column' We can only SELECT.

- DML Operations are not allowed. (common for all RDBMS).

create upper(ename) as 'u-name', sal*12 as 'anual' from emp;

↑ view based on function.

- View based on function, computed column, expression is allowed
- Can only SELECT.
- DML operations are not allowed.

DATE

```
create or replace view v1
as
select dname, ename from emp,dept
where
dept.deptno = emp.deptno;
```

↳ view base on join.

```
Select * from v1;
```

```
Insert into v1 values
('TRN','KING');
```

O/p

dname	ename
:	:

- view based join
- can only SELECT from above view
- DML operations are not allowed
- DML operations are allowed in Oracle with the help of 'INSTEAD OF' triggers

If you want to insert into a view the view has to be a simple view.

DATE

can we create a view based on view.? (view of view)

→ Yes.

→ it would be slower than a view based on table.

→ Where is it used ?

- We can normally have maximum 255 SELECTS with UNION operater.
- We can acheive union of >255 SELECTS with VIEWS.

→ Create a view V1 for 255 selects
Create a view V2 for 100 selects.
Creat a view V3 for V1 U V2.

Similarly,

- We can acheive >255 subqueries with VIEWS.

- We can simplify writing of complex queries.
e.g. join of 10 tables.

→ create a view V1 for first 5 joins
create a view V2 for next 5 tables join
Creat a view V3 for V1 join V2.

DATE

desc v1 ; → Show structure of view v1

Show tables ;

↳ Will show tables and views but it won't tell which is which;

To see which is a table & which is a view: -

Show full tables;

- There are 65 system tables in MySQL.
- The 65 tables that we see are not tables they are views!
- The main tables are hidden from us. (Hence views are used for security).

Show create view v1;

↳ Will show the SELECT stmt of v1 stored in system table.

DATE

```
use information_schema;
show tables;
```

→ Will show all the system tables (~~Actually they are system VIEWS~~)

```
show full tables;
```

→ Will show which one is table and which one is view.

→ After executing this, we will come to know about the fact that, All the system tables are VIEWS created by MySQL. (To hide the main table)

If we try to view any of these views, with:

```
show create view SYSTEM_VIEW_NAME;
```

→ Will give an error!

This is how the structure of main Sys. table is not revealed.

DATE

if you drop the table the indexes are dropped automatically.
If you drop the table the views remain.
If you drop the table, the views are invalid but they remain. you will have to drop the view separately

If you alter the table, you will have to recreate the views.

If you drop the table, the view is ~~invalid~~. not dropped
But if you create the table again in future, still the existing view is invalid ∵ view is still pointing to the old location of table,
∴ you will have to drop & recreate the views.

Uses of views:
- To restrict the access of users (GRANT)
- To restrict column access
- Restrict row access
- We can SELECT, INSERT, UPDATE, DELETE with the help of view
- In real life, entire application made on views.
- View with check option (used to enforce different restrictions on different users unlike 'chek' constraint that is common for all users)

- The SELECT Stmt of view is stored in DB in the 'compiled Format'
- View is an executable format of SELECT State ment.

DATE

- Hence, execution speed is very fast.
- Hiding source code from end user.
- Complex queries (union of 40 selects, complex joins) can be stored in view definition.
- To exceed the limits of SQL
  eg. union of more than 255 SELECTs
  Subqueries with more than 255 levels.

ADVANCED USES:

for
- Data mapping
- migration
- upgrades
- EAI (Enterprise Application Integration)
- EIM (Enterprise Integration Management)
- To convert 2D table into 3D table & vice versa
- To apply relational methods on Object table & vice versa etc.

DATE

(V. imp)

Normalization

- It is a concept of table designing.
- Normalization is dealing with,
  - What tables to create?
  - What will be the structure of table?
  - What will be columns?
  - What will be the datatypes?
  - " Width?
  - " Constraints)
- It is based on user-requirements.
- Normalization is a part of design phase.
- It is one of the phases of S/W development.
- It is a critical phase.
- Most of the time is spent on design phase (1/6)
- Why to do Normalization?
  - The aim of Normalization is to have an efficient table structure.
  - The aim is to avoid data redundancy. i.e. to avoid unnecessary duplication of data.
  - i.e same data should not be stored in multiple places.
  - DATA REDUNDACY leads to wastage of HD space. This is one of the aims of Normalization.
- The second aim of Normalization is
  - To reduce the problems of INSERT, UPDATE and DELETE.

DATE

- Over the years, SELECT stmt has become more powerful (due to joins, subqueries, views, clauses etc)...
- And there are many S/W available for data reporting like:
  - Oracle Reports / Graphics
  - Actuate
  - Seagate Crystal Report.

But INSERT, Update, DELETE has become more complicated. because businesses are getting complex

∴ Normalization will be done from an input perspective (What are the i/p to the S/W ? etc. questions need to ask the user).

↳ Questionaires are created to understand input requirements.

∴ Normalization is done from a FORM's perspective.

There are 9 steps for Normalization !

↳ If these steps are followed, anything can be Normalized.

DATE

An application, is divided into no. of Xactions like Register, place an order etc...

View the entire application on a per-transaction Basis and you Normalize each transaction SEPARATELY

- When user issues a commit, it becomes a Xaction.
- Entire application is made of TRANSACTIONS.
- You'll need to find out, all possible transaction and have to Normalize them.
  - using 9 steps.

e.g: in railway reservation, transactions are:

- Booking Ticket
- Canceling Ticket
- registration

9 steps of Normalization will be applied on each transaction

SELECT Stmt .. should not be considered as a Xaction.

e.g. (searching product in Amazon)

But buying a product is a transaction.

∴ SELECT doesn't change the DB state.

consider foll. transaction: ~~Ordering~~ Placing order.
(we will normalize this xaction)
Requirements of Amazon.

DATE

Form Terminology

Single row block

place order

cancel

multi row block

Repeatative fields

ONUM [5001]

CNUM [1001] CNAME [CDAC] CADDR [KHARGHAR]

CCITY [Mumbai] CPINCODE [400614] CEMAL [....]

ORDERDATE [30/10] DELIVERYDATE [2/11]

CMOBNO [ ] → 100

PRODCD	PRODNAME	QTY	RATE	ITEMTOTAL
P1	Apple Iphone 8plus	135	10	1350
P2	MARKER	100	5	500
P3	BATTERY	10	10	100

ORDERTOTAL [1950]

form.

(Before we start Normalization we draw a form)

∴ Normalization is done from Form's perspective.

ii) Ask the user for sample data.

Getting Ready for Normalization:-

a) Make a list of all the fields. for a given transaction.

ONUM
CNUM
CNAME
CADDR
CCITY
CPINCODE
CEMAIL
CMOBNO
ORDERDATE
DELIVERYDATE
PRODCD
PRODNAME
QTY
RATE
ITEMTOTAL *
ORDERTOTAL *

REPEATING GROUP

} computed colums.

PAGE

classmate

DATE

b) Strive for atomicity. (done in presence of user)

breaking into smaller parts.

eg: customer address (CADDR) can be broken down to

PLOT_NO,
BLDG_NO,
BLDG_NAME,
STREET,
AREA,
REGION,
LANMARK,
STATE etc.

Try to do this to all the fields.

c) Note down the type of data in each field (Datatype). (Done in the presence of user)

User doesn't tell you the datatypes of SQL.
he'll tell you about it in layman's terms like

e.g: user might tell you.

CPINCODE → numeric (6 digits)
CEMAILID → alphanumeric (15 chars)
.
.
ITEMTOTAL → QTY * RATE

DATE

(Done in presence of user)

d) List the properties for every field.

↳ e.g: consider mob. no: (CMOBNO).

CMOBNO → - 13 digits compulsory (including +91)
- is update allowed ? (yes, user can update his no).
- is delete allowed ?
- is insert allowed ?
- is it an enterable / non-enterable field. (~ textarea)

⋮

like these properties, there are total ~~343~~ 304 properties of each field.

- Make questionaires to get these properties from user & ask him.

e) Get user signoff.

(After this, we wont need user for feasther steps)

After user signoff, user involvement ends

DATE

f.) For all practical purposes, we can have a single table for all these columns.
(put every column in one table).

g) Assign a suitable table name.

↳ in our case : CUSTOMER_PLACES_AN_ORDER

h) Assign data types

↳ we have noted type of data fields from uses in step c)

- if you observe ITEMTOTAL & OTOTAL are computed columns.

i.e ITEMTOTAL → qty * RATE

OTOTAL → SUM(qty * RATE)

i) Remove the computed columns

∴ Remove ITEMTOTAL & OTOTAL.

∵ computed columns are not a part of Table.

DATE

(unique)
j> KEY element will be primary key of this table.

∴ The table is storing the order details of customers for this Xaction (placing order.).

∴ ONUM is primary key.

if you cant find primary key, look for composite PK.

If composite PK is not found, Add a Surrogate key!

AT THIS POINT DATA is IN UN-NORMALIZED FORM (UNF)

↳ UNF is the starting point of Normalization

***** NORMALIZATION BEGINS from HERE **********

i> Remove the Repeating group into a new table.

CUSTOMER_PLACES_AN_ORDER

ONUM
CNUM
CNAME
CADDR
CCITY
CPINCODE
CEMAILID
CMOBNO
ORDERDATE
DELYDATE

Repeating group is itself 1 to 0 table. (See the form)

PRODCD
PRODNAME
QTY
RATE

↑ New table for Repeating group.

classmate

PAGE

DATE

ii) Key element will be primary key of new table.

ONUM	PRODCD
CNUM	PRODNAME
CNAME	QTY
CADD	RATE
⋮	
DELYDATE	

iii) (This step may or may not be required)

~~In above~~ (New table)

In above table PK is PRODCD. But the above table will store Product info. of orders, but, ∵ we are Normalizing Order Xaction, multiple orders can have same prod. it will result in duplicacy ∴ We look for a composite key.

When you'll place an order a new row will be inserted in both ~~the~~ tables ~~as~~ shown above. So, if there's a same product in 2 orders, it will be inserted only once (∵ PRODCD is PK) in 2nd table Add the primary key of original table to the new table to give you a composite primary key.

ONUM	ONUM
CNUM	PRODCD
CNAME	PRODNAME
⋮	QTY
DELYDATE	RATE

← Will now uniquely identify each product as per the order.

~~W~~ There's no way of identifying that the product belongs to which order. ∴ we need composite key

classmate PAGE

DATE

Repeat the above 3 steps infinitely till you cannot Normalize any further.

↳ After doing this we'll get FIRST NORMAL FORM. (FNF) (1NF).

Imp ★ FIRST NORMAL FORM -> Repeating groups are removed from table design.

1: Many Relationship is always encountered here.

eg:

- dept ↓ emp -> one dept has many employees.
- one order may have many products.

iv) Only the tables with composite primary key are examined only

composite P.K { ONUM, PRODCD } key elements (part of PK)

Non-key elements { PRODNAME, QTY, RATE }

purpose of PK is row uniqueness.

- PRODNAME is dependent on PRODCD. it has nothing to do with ONUM.
- RATE ..... PRODCD ...
  .... ONUM

- QTY is totally dependent on both ONUM & PRODCD.
  ∴ QTY deserves to be in the table.

PAGE

classmate

DATE

∴ Out of the non-key elements, those fields that are not dependent on the entire (all the key elements) composite primary key, they are to be removed into a new table.

CUSTOMER_PLACES_AN_ORDER

ONUM

PRODCD

QTY

PRODNAME

RATE

↑

New table.

Now, the new table needs a primary key.

vi) The key element on which the columns of new table were originally dependent (PRODCD), it is to be added to the new table, and it will be the primary key of new table.

perform v) & vi) repeatedly unless you can not normalize any further. → 2NF

ONUM

PRODCD

QTY

PRODCD

PRODNAME

RATE.

DATE

SECOND NORMAL FORM (SNF) (2NF):

Every column is functionally dependent on the Primary key. This is known as functional dependency.

Functional dependency means, W/O Primary key, that column cannot function.

↳ QTY is fully dependent on PK (DNUM, PRODCO)

PRODNAM, RATE are dependent on PK (PRODCO)

↳ i.e. functional dependency is encounted in 2NF

DATE

vii) Only the NON-key elements are examined

Non-key elements - Not a part of PK.

ONUM	ONUM	PRODCD
CNUM	PRODCD	PRODNAME
CADDR	QTY	RATE
CCITY		
CPINCODE		
CEMAILID		
CMOBNO		
~~ORDERNO~~		
ORDERDATE		
DELYDATE		

Not dependent on PK (ONUM)

dependent

dependent on PK (ONUM, PRODCD)

both depend on PK (PRODCD)

viii) Inter dependent columns are removed into a new table.

ONUM	CNUM	ONUM	PRODCD
ORDERDATE	CNAME	PRODCD	PRODNAME
DELYDATE	CADDR	QTY	RATE
	CCITY		
	CPINCODE		
	CEMAILID		
	CMOBNO.		

↑ New table.

iv) key element will be primary key of new table, and the primary key of new table, it is to be retained in the original table for relationship purposes.

ONUM	CNUM	ONUM	PRODCD
ORDERDATE	CNAME	PRODCD	PRODNAME
DELYDATE	CADDR	QTY	RATE
~~CA~~	CCITY		
CNUM	CPINCODE		
	CEMAILID		
	CMOBNO		

Do this repeatedly unless no further Normalization Can be done. ~~We~~ After o doing this, We acheive (3NF).

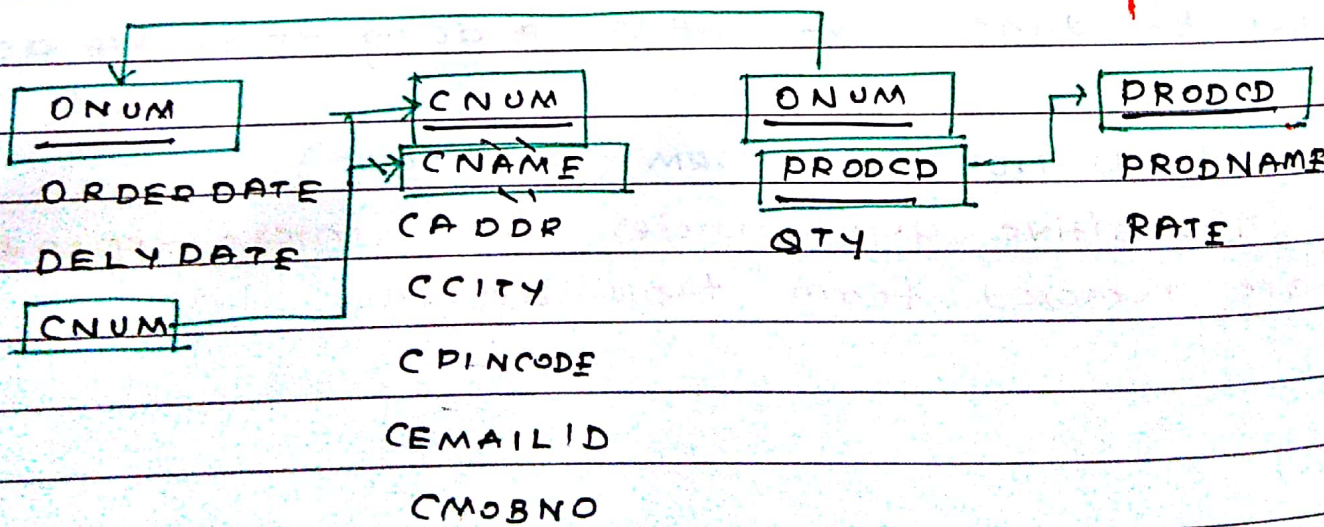
THIRD-NORMAL-FORM (TNF) (3NF)

Transitive dependencies are (inter-dependencies) are removed from table design.

Give Suitable Name to all the tables.

ORDERS	CUSTOMERS	ORDER_DETAILS	PRODUCTS
ONUM	CNUM	.	.
ORDERDATE	CNAME	.	.
CNUM	CADDR	.	.
	CCITY		
	CPINCODE		
	CEMAILID		
	CMOBNO		

- PRIMARY KEY is a by-product of Normalization.
- We already have a list of constraints from user (CK Step)
- only one constraint we have to take care of: → FOREIGN KEY.



DATE

4th Normal Form (4NF)

- AKA Boyce-Codd Normal Form (BCNF)
- It is an extension to 3NF
- You may or may not implement 4NF
  - It has nothing to do with (duplication, data efficiency & INSERT, UODATE, DELETE).
- You may want to use 4th Normal Form to reduce the possibility of data errors at the time of insession.

eg: (e.g. of singpore)

CNUM	CNAME	.....	~~CCITY~~ ~~CADDR~~	CPINCODE	.....
1001			KHARGHAR	400614	
1002			Nariman pt	400021	
1003			pune	411037	
1004			Singapore	400614	...

REFER NEXT PAGE

This is an error in data (Logical).

for Amazon, this is the most important information for an order. Amazon would not care for your name, email or mob no.

The solution to this is, Creating a separate table for such important fields.

∴ CPINCODE & CADDR are kept in a separate table (CPINCODE - CCITY).

3NF - BCNF → For OODBMS

DATE

CNUM
CNAME
CADDR
CCITY
CPINCODE
CEMAILID
CMOBNO

CPINCODE → CCITY

CPINCODE	CCITY
400614	Mumbai
400021	mumbai
411037	Pune

CNUM	CNAME	CADDR	CCITY	CPINCODE	...
1001	CDAC	KHG	MUM	400614	...
1002	AITYCP	Nariman pt	MUM	400021	...
1003	SUNBEAM	Market yard	PUNE	411037	...
1004	CYBAGE	Tapri	Singapor	400614	..

CPNCODE_CCITY

CPINCODE	CCITY
400614	MUM
400021	MUM
411037	PUNE

∴ At the time of registration, Amazon, first will ask for pincodes.

DATE

De- Normalization

itemtotal = qty * rate

but qty & rate are in diff tables after 4NF.
∴ join is needed to cal. itemtotal.

similarly

ORDERTOTAL = sum (qty* rate)

...

...

These are computed columns.

If the data is large, if the SELECTs are slow, add an extra column to the table to improve performance, to make the SELECT stmt work faster.

- Normally done for computed columns, expressions, function-based columns etc.

∴ the computed columns are stored in appropriate columns.

ORDERTOTAL is added to table ORDERS & ITEMTOTAL is added to ORDER_DETAILS.

ORDERS	...	ORDER_DETAILS	...
ONUM	.....	ONUM	
ORDERDATE		PRODCD	
DELYDATE		QTY	
CNUM		ITEMTOTAL	
OTOTAL			

D:77 L:9

DATE 31 10 2017

MySQL PL

Is very easy language

- MySQL Programming lanonguage.

PL/SQL

- If you are using Oracle RDBMS, we have
- PLSQL as a programming Language.
- it stands for Procedural Language SQL.
- It is (PLSQL) a #1 DB programming language in the world (commercially)
- PLSQL is #2 DB programing lang in the world (overall)

MySQL PL

- It is a programming language of MySQL.
- It is used for data base programming..
- Every lang has a purpose.
- C is gen. purpose lang. (
- C++ is sys prog.
- HTML is for web purposes.
- MySQL is for DB programming

↳ e.g. for calculating HRA (HRA_CALC)
TAX_CALC, ATTENDANCE_CALC etc.

- It is used for server side data processing.
- Some processing that has to be done on Server.

e.g:

STORED PROCED.

DATE

If you want to do TAX-CALC on table with 10000 rows. Such processing is done on SERVER RAM to reduce N/W traffic.

- The MySQL program can be called in MySQL Command Line client.
- We can also call it in MySQL workbench.
- You can call it in PHPMyAdmin.
- You can call it through Oracle Reports, Oracle Menus, Oracle Graphics, Java, .Net, etc.

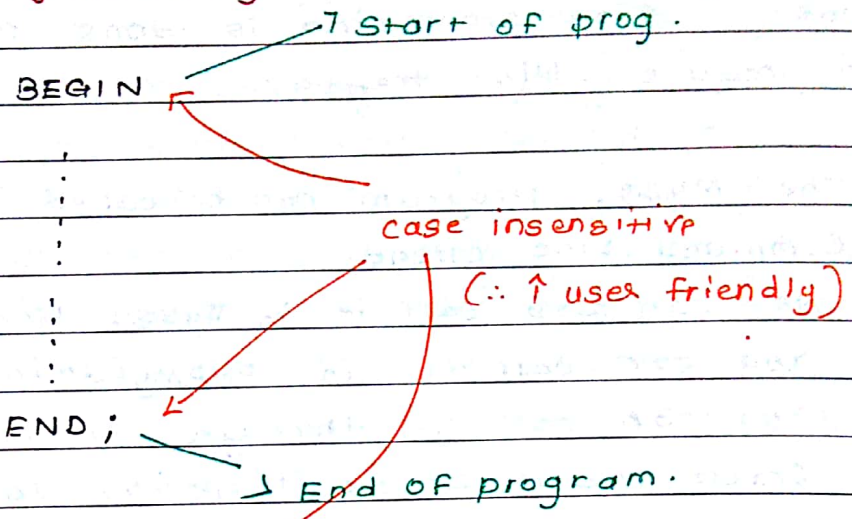
-:. It can be called through any front end S/W

To call it from java → JDBC drives
.Net → ODBC drives

MySQL PL has 4GL features (4th gen lang.)
- it is not completely object oriented but it supports few 4GL Features.

DATE

Your MySQL program will look ~~like~~ like:



MySQL Block

```
BEGIN
Insert into DEPT values (1,'a','B');
END;
```

Very easy Syntax

→ This is a program to add a row to DEPT table.

Open source.

- MySQL PL is the most popular DB programming language in the world. (42%.)
- PL SQL is #1 (commercially) 20%. (is mongoDB)

temaining 38%. is others.

- PL SQL is 63%. (in commercial, 2/3)

DATE

- A mysql prog. is a block of commands.
- A mysql prog is refered as 'MySQL Block'
- (use text editor or source code editors (Atom) for programming)

Parent block

```
BEGIN
    :
    BEGIN        <- child block
        :
    END;
    :
END;
```

- We can have a block within a block.
- ∴ Nesting of block is Allowed.
- No upper limit for Nesting.

∵ We can have a block within a block, MySQL PL is called 'BLOCK LEVEL LANGUAG'.

- The execution will always be top to bottom.

Why to do Nesting of blocks? ↑ organized

↳ For making it modular. (i)

- i.e. Separate functionality in each block.
- it becomes easy for reading, writing, debugging, maintaining.

- It is recommended that, a block should not exceed 25 lines. (Not compulsory but is an industry std.).

↳ As per SEI & CMM (~ to ISO)

SEI -> S/W Engineering Institute.

CMM ->

DATE

ISRO is a level-5 company.

ISO is a general standard.

SEI is a specifically standard for S/W.

SEI-CMM -> is a standard for S/W companies

Std ranges from (Level 1 to 5)

lowest std

Highest Std.

- If a company wants to have level 5 std, it will approach SEI-CMM
- SEI-CMM will give a set of guidelines (rules) & standards.
- After 2-3 years SEI-CMM performs an audit for that company. (Going through each & every prog.)
- If the company satisfies these guidlines & standards, & App level 5 standard is given to the company.
- Every level has a diff set of guidelines & standards.

- There's no upper limit on level of Nesting as per the documentation.
- But it is 8095 blocks.

- Data Hiding (ii)

```
BEGIN
   :
   (x) ←var
   BEGIN
   :
   (y) ←var.
   END;
END;
```

- Scope of y is only the inner block.
- x is available to both the blocks.

∴ We can control the scope of variables.

- ∴ => Data hiding can be acheived.

- Efficient Exception management. (iii)

- If one block is failing, it should not fail the Whole prog.

∴ manage (Handle) Exceptions in each block.

```
BEGIN
   :
   BEGIN
      ;  ← Handle Exception of this block here only
   END;
END;
```

eg: If GST_CALCULATE block fails it should not fail other blocks like 'SERVICE_TAX_CALC'.

- Screen i/p and screen o/p are not allowed (scanf, printf etc. Not available)
- MySQL PL is only used for processing!

```
BEGIN
  :
    BEGIN
      :
     SELECT * from emp;
      :
    END;
  :
END;
```

We can use SELECT stmt inside a block, but it is not recommended!

SQL commands that are allowed in MySQL PL:

- Create
- alter
- drop
- insert
- updat
- delet
- rollback
- Commit,
- Savepoint
- Select.
  - Allowed but not recommended.

DATE

Select stmt is not recommended but it is necessary & useful for subqueries.

```
BEGIN
   :
     BEGIN
       :
       delete from emp where deptno =
          (select deptno from emp where
                ename = 'Thomas');

     END;
END;
```

- DCL commands (GRANT & REVOKE) are not allowed inside MySQL PL.

∴ We cannot print on screen. then how to see the o/p of our program.

↳ - Create an Output table before.
- put (insert) the processed o/p in this table.
- We can see the o/p from this table.

Output table

This table will act like an output table for our MySQL program.

```
Create table tempp
(
    fir int(4),
    sec char(15)
);
```

The table will store o/p of MySQL PL program.

TEMPP

FIR	SEC
:	:

Can have any no. of columns. depending on the program & the type of o/p of that program.

NOTE:

MySQL PL program is written in the form of Stored procedure.

Stored in DB.

DATE

STORED PROCEDURE

- It is another Stored Objects.
- Stored objects are the objects stored in DB. create tables, indexes, views are e.g. of Stored objects.

→ The entire code of a procedure (prog) is there in system table (DB)

A procedure is a routine. It is a set of commands that has to be called explicitly

- A procedure does not RETURN anything.
- It is like a void function of C/C++.

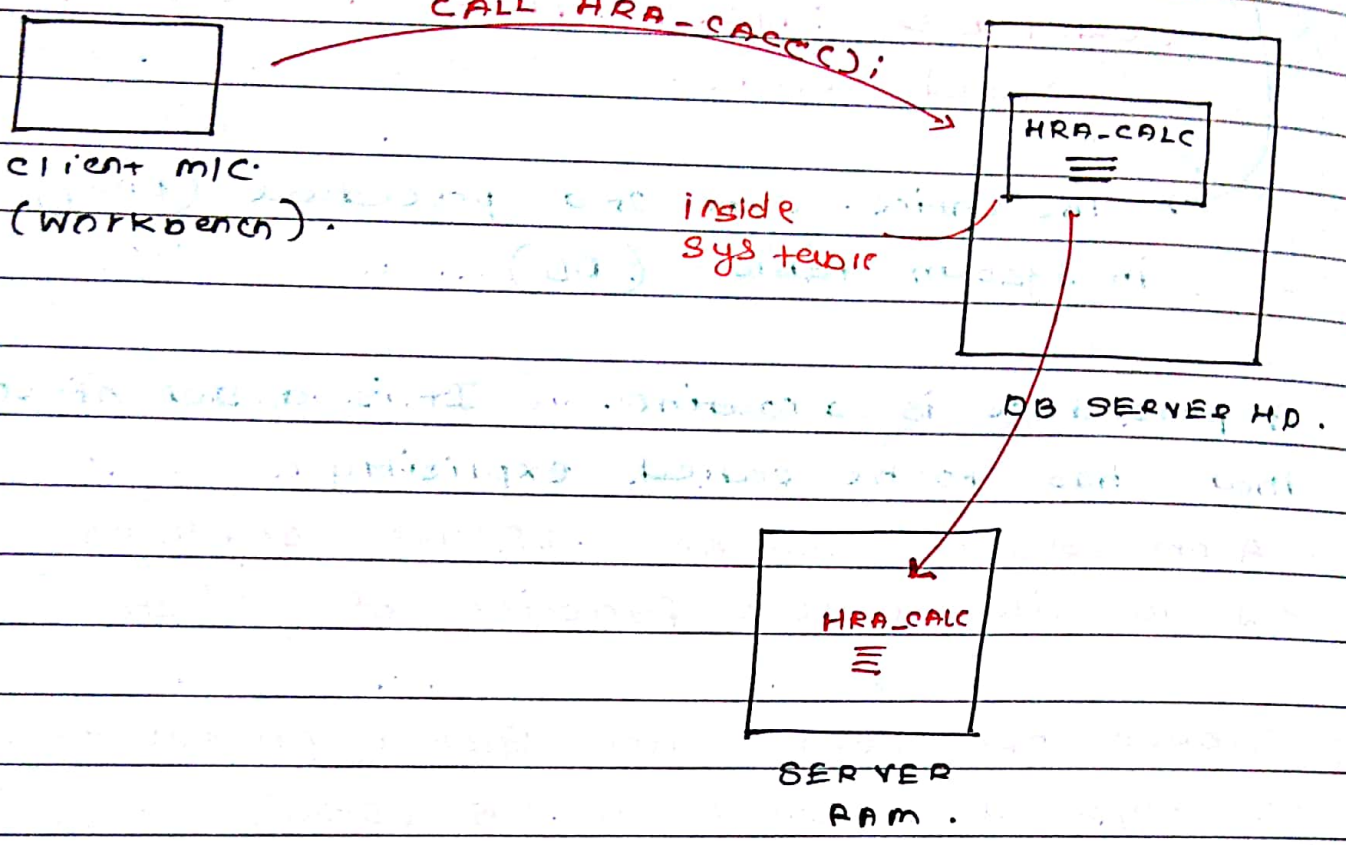
- STORED PROCEDURES are Global procedures.
  ∵ they are stored in DB. (server.)
  we can access it any where.

∴ Once we create a procedure, we can call it from any where.

→ Can be called in MySQL Command Line Client, MySQL Workbench, Oracle Forms, Oracle Reports, Oracle Menus, Oracle Graphics, Java, .Net etc.

- It can be called through any Front end S/W

There is a Sys. table for procedure.
The entire procedure is stored inside a table.



- The Exen of procedure takes place in server ram.
- When a procedure is called, the copy of procedure is brought to RAM, it is exed & after exen it is gone.

- The code of procedure is stored in system tables is in the COMPILED FORMAT.

↳ ∴ Exen is very fast
& code is Hidden from user.

DATE

- Stored procedure can have local variables.
- Within the stored procedure, we can have IF statement, LOOPS, etc.
- one stored procedure can call another S.procedure. (just like one fn can call other fn.)
- Stored procedure can call itself. (Recursion).
- Stored procedure can have parameters.
  e.g. For HRA_CALC(), we can pass SAL as parameter.
- OVERLOADING of Stored procedure is Not allowed.
  - You cannot create 2 Stored procedure with the same name no matter if the no. of parameters passed is different or the data type of parameters is different.

DATE

Creating a procedure (Begining MySQL P2)

```
Create procedure abc()
begin
    insert into tempp values (1, 'Hello');
end;
```

procedure name (~~show~~ Has to be unique)

This is Source code of procedure.

First we will create a proc.
later we will call the procedure

';' indicates end of command.

→ it is a default delimiter

Boundary of procedure

```
delimeter //
Create procedure abc()
begin
    insert into tempp values (1, 'Hello');
end; //
delimeter ;
```

indicates that it is (procedure) is a one unit.

defining delimiter back to ';' for feather commands.

Go to mySQL Work bench → copy the procedure these 4 execute it (ctr + Enter)

→ With this, the source code of abc() will be stored in Sys. table in Server HD. (permanantly. Not for a session)

DATE

```
call abc();
```

(Ctrl + Enter)

Wont show any o/p, will execute in background (server RAM).

procedure abc() is called from servlet as explained befor.

abc() will insert (1, 'Hello') in tempp (o/p table) & will return back.

```
Select * from temp;
```

Now you can see the o/p of abc().

o/p

FIR	SEC
1	Hello

You can drop procedure later if you want

```
drop procedure abc;
```

```
delimiter //
create procedure abc()
begin
    insert into tempp values (1, 'Hello');
end; //
delimeter ;
```

this ';' is used within a block. It indicates end of command WITHIN the block

It is optional.

// indicates end of the procedure

Why to do it?

When we are having BEGIN & END

(compiler is not intelligent.)

it will (in case of Nested blocks).

```
Create proc...
  Begin
    :
      Begin
        :
      End ;
    :
  End;
```

DATE

In MySQL Workbench, we have 2 compilers

a) MySQL SQL compiler
b) MySQL PL ~~SQL~~ compiler.

default delimeter is ;

↳ it indicates end of command.

When mysql SQL compiler encounters ;

e.g select * from emp;

↓ it will

i) read
ii) compile
iii) plan
iv) execut.

default delimeter is ;

When the MySQL PL compiler encounters ;

↓ it will

i) read it as ONE UNIT (one procedure).
ii) compile.
iii) plan
iv) Store it in DB in compiled format.

later we can call it using ~~abc();~~ call abc();

DATE

```
create procedure abc()
    begin
        insert into tempp values (1, 'Hello');
        insert into tempp values (2, 'Hi');
        commit;
    end;
```

Now when MySQL PL compiler encounters ';' it will assume it as a one unit (a procedure) It wont read any further commands

So, only 1st insert will be a part of procedure abc(),

it will compile it & store it in the DB.

One invoking

```
call abc();
```

↳ will give an error.

('END' Not found in procedure)

DATE

variable declaring & initializing

delimiter //

```
create procedure abc()
begin
    declare x int(4);
    set x = 10;
    insert into temp values(x, 'Hello');
end; //
```

Variable declaration

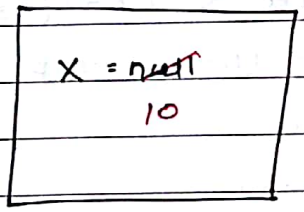
Assignment operator.

delimiter ; ← optional.

- The scope of x is within the block.
- it is a local variable.
- It is killed when block is ENDed.

- 'x' variable is created in server RAM with null value.

- ∴ When you declare a ~~value~~ variable w/o initializing it, it will store a null value. (variable of any DT).



Server RAM

TEMP

FIR	SEC
10	HELLO

```
call abc();
select * from temp;
```

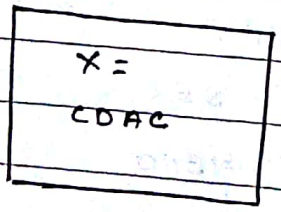
→ O/P

FIR	SEC
10	Hello

```
delimiter //
  create procedure abc()
    begin
        declare x int(4) default 10;
        insert into tempp values(x, 'Hello');
    end ; //
delimiter ;
```

You can declare a variable and assign a value simultaneously.
(Recommended ! -> Saves one line)

```
delimiter //
  create procedure abc()
    begin
        declare x char(15) default 'CDAC';
        insert into temp values(1, x);
    end ; //
delimiter ;
```



TEMPP

FIR	SEC
1	CDAC

DATE

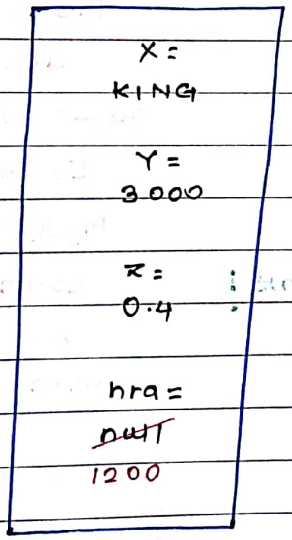
Calculating HRA

```
delimiter //
   Create procedure abc()
   begin
        declare x char(15) default 'KING';
        declare y float(7,2) default 3000;
        declare z float(2,1) default 0.4;
        delare hra float(7,2);
        Set hra = y*z;

        insert into tempp values(y,x);
        insert into tempp values(hra, 'HRA');
   end; //
delimiter;
```

Hard coded.

Assuming 'FIN' col^n in TEMAP Table is Float.

Assuming hra is 40%. of sal.



TEMPP

FIR	SEC
3000	KING
1200	HRA

DATE

passing parameters

```
delimiter //
create procedure abc( x char(15), y float(7,2), z float(2,1))
begin
    declare hra float(7,2);
    set hra = y * z;

    insert into temp values(y,x);
    insert into temp values(hra, 'HRA');
end; //
delimiter ;

call abc('KING', 3000, 0.4);
call abc('SCOTT', 2000, 0.3);
select * from tempp;
```

inserting only 2 values (∵ TEMPP has only 2 columns)

o/p

TEMPP

FIR	SEC
3000	KING
1200	HRA

To you can use this select inside abc(), to get result directly on calling. It will work with MySQL workbench, command line client but wont work with java, html etc.

```
-- For single line coments!
/* for
   multi-line
      comment */
```

Write as many comments as you can.

ACC, to a CMM standard, there should be at least 1 comment for every 2 statements. (33%-50%)

ideally -> 1 comment for every statement.

∵ Program is written once but read several times.

DATE

```
delimiter //
create procedure abc(...)
begin
    .
    .
    .
    select * from temp;
end; //
delimiter ;
```

→ o/p comes on screen

```
call abc('KING', 3000, 0.4);
```

↑ We will not need to write select * from temp after this.
∵ it is ~~ex~~ written inside abc().
∴ After execution, abc will show the tempp table on screen.
It will work in MySQL Workbench, Command line client but won't work with java, html etc.

DATE

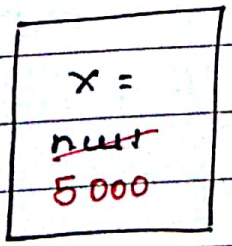
Accessing values from existing table.

EMP

ENAME	SAL	JOB
SCOTT	3000	CLERK
KING	5000	MANAGER

x ~~should~~ Must have same datatype & width of sal column of EMP. Or else, will get a 'value error' at runtime

```
delimiter //
  create procedure abc()
  begin
      declare x int(4);
      select sal into x from emp where ename='KING';
      /* processing, e.g. hra = x * 0.4, etc */

      insert into tempp values (x, 'KING');
  end;//
delimiter ;
call abc();
select * from tempp;
```



TEMPP

FIR	SEC
5000	KING

DATE

Avoid altering the table (column). Stored procedures may not work.

e.g if we alter the SAL column to float, the stored procedure will not work.

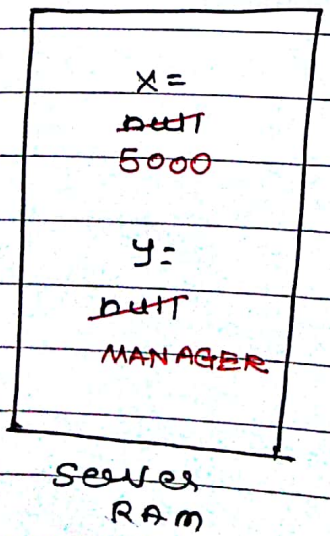
- what if there are 2 employees with same name (KING)
- procedure will give a runtime error.
  'Too many rows found'.

Try to
∴ select on basis of unique cols.

DATE

Accessing multiple values from existing table.

```
delimiter //
create procedure abc()
' begin
      declare x int(4);
      declare y char(15)

      select sal, job into x, y from emp
      where ename = 'KING';

      /* processing
            eg. hra = x * 0.4,
                lower(y)
      */

      insert into tempp values(x,y);
end; //
delimiter ;

call abc();
select * from tempp;
```



TEMPP

FIR	SEC
5000	MANAGER

DATE

To see which all procedures are created,

```
Show procedure status ;
```

↳ Shows all procedures in the database

```
Show procedure status where db = 'DAC1' ;
```

↳ All procedus from 'DAC1' schema.

```
Show procedure status where name like 'A%' ;
```

↳ All the procedures starting with 'A'.

~~If you want to modify the procedure.~~

To view source code of the stored procedure.

```
Show create procedure abc ;
```

↑ To recover the source code.

DATE

In multi-user env., one user can share its procedures with other users (GRANT)

'DAC1' → abc() ← 'DAC2'

(DAC1 is the owner of abc() ∵ DAC1 has created the procedure.)

DAC1 can share abc() with DAC 2

DAC1_MySQL> create procedure abc()..... ;

DAC1_MySQL> grant execute on procedure abc to DAC2;

DAC2_MySQL> call DAC1.abc()

Schema . procedurename.

To revoke:

DAC1_MySQL> revoke execute on procedure abc from dac2;

Assignment

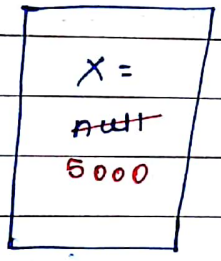
MySQL Exercise 1

DATE

MySQL PL : If Statement (Decision Making)

```
delimiter //
  create procedure abc()
    begin
      declare x int(4);
      select sal into x from emp where ename='KING';
      if x>4000 then
          insert into tempp values(x,'High sal');
      end if;

    end;//
delimiter;
Call abc();    select * from tempp;
```

Procedure always executes in server RAM.



Server RAM

TEMPP

FIR	SEC
5000	Highsal

```
if condition then
      .....
      .....
end if;
```

Assuming that KING's sal = 3000

∴ x = 3000

the if condition will not be satisfied

∴ O/p will be nothing.

```
delimiter //
 create procedure abc()
   begin
      declare x int(4);
      Select sal into x from emp where ename='KING';

      if x>4000 then
         insert into temp values(x,'High sal');
      else
         insert into temp values(x,'LOW sal');

      end if;
   end ;//
delimiter ;

call abc();
Select * from tempp;
```

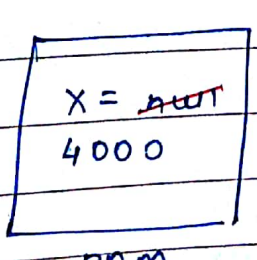
O/P

FIR	SEC.
3000	low-sal

DATE

We can have nested 'if':

```
//delimiter
create procedure abc()
begin.
    declare x int(4);
    select sal into x from emp where ename='KING';
    .
    if x>4000 then
        insert into temp values(x,'High sal');
    else
        if x<4000 then
            insert into tempp values(x,'Low sal');
        else
            insert into tempp values(x,'Mediumsal');
        end if;
    .
    end if;
end;//
delimiter;

call abc();
select * tempp;
```



o/p

TEMPP

FIR	SEC
4000	Mediumsal

DATE

elseif Statement

```
delimiter //
create procedure abc()
  begin
    declare x int(4);
    select sal into x from emp where ename='KING';

    if x > 4000 then
      insert into temp values (x,'High sal');
    elseif x < 4000 then
      insert into temp values (x,'Low sal');
    else
      insert into temp values (x,'Medium sal');
    end if;

  end ; //
delimiter ;

call abc();
select * from temp;
```

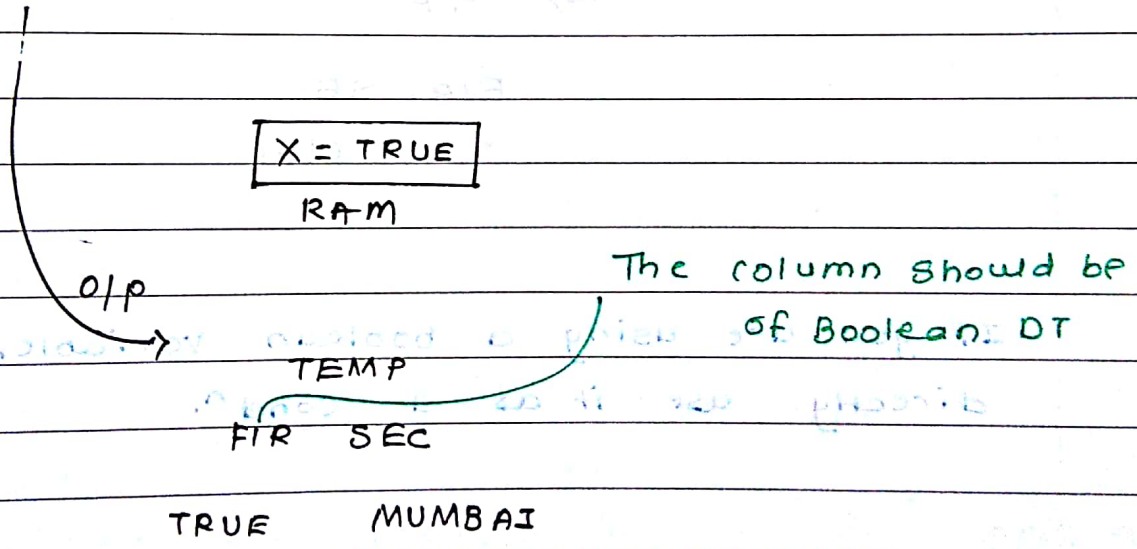
→ Exactly like default else of c/c++/java.

→ O/p.

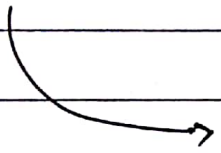
FIR SEC.

```
delimiter //
  create procedure abc()
    begin
        declare x boolean default TRUE;

    if x then
            insert into values (x, 'Mumbai');
        end if;

    end; //
delimiter ;

call abc();
select * from temp;
```

case insensitive



```
delimiter //
  create procedure abc()
    begin
      declare x boolean default FALSE;
        if not x then
          insert into tempp values (1,'Delhi');
        end if;
    end;//
delimiter ;

call abc();
select * from temp;
```



O/P

FIR	SEC
1	Delhi

If you are using a boolean variable, you can directly use it as a condⁿ.

DATE

CASE Statement

- It is faster than if statement ; (why?)

```
delimiter //
    create procedure abc()
    begin
        declare x int(4);
        select sal into x from emp where name='KING';

        case
            when x > 4000 then
                insert into tempp values(x,'High sal');
            when x < 4000 then
                insert into tempp values(x,'Low sal');
            else
                insert into tempp values(x,'Medium sal');
        end case;
    end; //
delimiter ;
```

default case.

- Use 'case' statement wherever possible. If a problem cannot be solved by CASE statement then go for 'IF' statement

- Limitation of case is that Nesting is not possible.

- else (default else) is optional. BUT
- if else is not provided & if none of the cases are satisfied, then MySQL will give an error message.

DATE

- If you want to skip the statements inside 'else' and avoid the error message, you supply 'else' with an empty Begin & end block.

```
:
:
else
    begin
    end;
end case;
:
:
```

↳ to static var:

Session Variables:

- Session variables are Global variables.
- Once you create a session variable, it will remain in the server RAM till you exit / disconnect.
- Session variables can be accessed at the command prompt of MySQL command line client, MySQL Workbench, stored procedure, front end s/w etc.

```
set @x = 10;        /* creating a session var */
select @x from dual;
```

↳ o/p
10

Will be int. automatically ∵ assigning '10'

Session var → As a counter
Session var → As a ROWNUM.

classmate

DATE

Loops

Note (about session var).

as a counter.

```
Set @num_of_inserts = 0
insert ..... ;
set @num_of_inserts = @num_of_inserts + 1 ;
```

can be accessed from any where. (different procedures)

Used for Repeatative / iterative processing.

While loop.

- We always check for some condition before entering the loop

```
While expression DO
  ...
  ...
END while ;
```

lets assume there are 5 rows in emp table.

```
Set @x = 0
select @x+1 from emp ;
```

o/p:

1
2
3
4
5

classmate MySQL has no feature of ROWNUM PAGE
with this, we can make a it work like ROWNUM.

DATE

```
delimiter //
  create procedure abc()
    begin
      declare x int(4) default 1;

      While x<10 do
        insert into tempp values(x, 'inloop');
        set x= x+1;
      end while;

    end; //
delimiter ;

call abc();
select * from tempp;
```

~ x++; of c/c++/java.

↓

doesn't work in SQL

↳ O/P

TEMPP

FIR	SEC
1	in loop
2	in loop
3	in loop
⋮	⋮
9	in loop

What will happen if there's an infinite loop?

- if we comment 'set x = x + 1;', the procedure will get into an infinite loop.
- The application will stop responding. We can kill the task (application) & restart it.

BUT

- The procedure will still be executing in the Server RAM ~~fe~~.
- This will fill up the SERVER HD completely. & Server will CRASH.

Repeat Loop : (similar to do while loop).

↳ Will execute ~~only once~~ at least once.

```
REPEAT
    .....;
    .....;
UNTIL expression
END REPEAT
```

```
delimiter //
  create procedure abc()
    begin
        declare x int(4) default 1;
        REPEAT
            insert into tempp values(x, 'in loop');
            set x = x + 1;
        UNTIL x > 5 ;;
        END REPEAT;
    end;//
delimiter ;

call abc();
select * from tempp;
```

semicolon is not there!

↳ O/P

TEMPP

FIR	SEC
1	in loop
2	in loop
3	in loop
4	in loop
5	in loop

DATE

If we initialize x to 100, the loop will be executed once.

Loop, Leave and Iterate Statements

- Leave statement allows you to exit the loop (similar to break; statement)

- Iterate statement allows you to skip the entire code below it and start a new iteration (Similar to continue; statement).

- Loop statement executes a block of code repeatedly with an additional flexibility of using a loop label.
  - ↳ Name given to a loop. (label)

DATE

```
delimiter //
create procedure abc()
begin
    declare x int(4) default 1;
    pqr_loop : loop
        if x > 10 then
            leave pqr_loop;
        end if;
        set x = x + 1;
        if mod(x, 2) != 0 then
            iterate pqr_loop;
        else
            insert into tempp values (x, 'insideloop');
        end if
    end loop;
end;//
delimiter ;

call abc();
select * from temp;
```

O/P

TEMP

FIR	SEC
2	inside loop
4	inside loop
6	inside loop
8	inside loop
10	inside loop

X = 1 ~~2 3 4 5 6 7 8 9 10~~ 11

Sub-blocks (Nested blocks)

- Block within Block. (it is a feature of OOPs)

```
delimiter //
    create procedure abc()
    begin
        declare x int(4) default 1;
        insert into temp values(x,'before sub');

        Begin                          Local var.
            declare y int(4) default 2;
            insert into tempp values(y,'in sub');
        END;

        insert into tempp values(x,'after sub');
    end; //
delimiter;

call abc();
select * from tempp;
```

O/P

TEMPP

FIR	SEC
1	before sub
2	in sub
1	after sub.

Data hiding (encapsulation)

Main block cannot access variables of the subblock.
e.g. if we try to access 'y' outside inner block → error. But reverse is allowed! Sub-block can access Var of main block (outer).
i.e 'x' can be accessed inside inner block.

- Sub block can change the value of the main block's variable. & it will affect the remainder of program.

```
:
:
:
insert into tempp values (x, 'before sub');
Begin
    declare y int(4) default 2;
    set x = x+y;
    insert into tempp (value x, 'in sub');
End;
:
:
```

if the name of outer block var = name of inner block var, higher priority is given to local variable

```
delimiter //
create procedure abc()
begin
    declare x int(4) default 4;
    insert into temp values (x, 'before sub');
    Begin
        declare x int(4) default 3;
        insert into temp values (x, 'in sub');
    End;
:
:
```

DATE

```
set @x = 1;
delimiter //
    create procedure abc()
        begin
            declare x int(4) default 1;
            insert into tempp values(x, 'beforesub');
            Begin
                declare x int(4) default 3;
                insert into tempp values(@x, 'in sub');
                insert into tempp values(x, 'in sub');
            End;
            insert into tempp values(x, 'after sub');
    end; //
delimiter ;

call abc();
select * from tempp;
```

o/p

TEMPP

@x	FIR	SEC
	1	before sub
	1	in sub
	3	in sub
	1	after sub.

Assignments:

MySQL Exercises 1, 2, 3.

← lab exam.

- Create table with constraints
- insert rows
- 5 - 10 SELECTs
- 1 stored procedure with cursor
- 1 stored function

classmate

PAGE

L:10 D:76

DATE 01/12/2017

MySQL Cursors

- Cursors are present in all RDBMS or some DBms
- cursors are present in some frontend also.
  (.Net)

In DB Server HD

EMP

int(4) EMPNO	varchar(5) ENAME	int(4) SAL	int(2) DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

TEMPP → o/p table.

FIR	SEC

- The programs are executed in server ram.
- A cursor is a variable.
- ∵ it is a variable, cursor is in Server RAM.
- All the variables are in server RAM.
- So, what is the difference betn var & cursor?

```
declare x int(4)
```

x = null

RAM

```
select sal into x from emp where
    empno=1;
```

x = ~~null~~ 5000

```
set hra = x * 0.4
```

↳ calculating hra for first row.

classmate

DATE

```
select sal ..... empno = 2 .
set hra = x * 0.4 ;
select . . .
. . . . .
select . . .
. . . . .
select . . .
. . . . .
```

3

4

5

OR we can put it in the WHILE loop.

but still

↑ calculating hra for all employees.

- I/O will be performed 5 times!

↳ I/O means, for each select, a row will be fetched FROM HD to RAM

Imagine if your table has over a million rows.

DATE

- Cursor can store multiple rows.
- Cursor is used for storing multiple rows.

Cursor PGR

EMPNO	ENAME	SAL	DEPTNO
1	...	....	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...

- It is like a copy of the table stored in a variable.. (in RAM)
- It is similar to 2D R. Array.
- :. If you have large no. of rows in HD, you should have large no. of RAM

Cursor P PGR

1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

- Now, we can directly work on the cursor, (in RAM). ~ (Like paging)

command to create a cursor.

declare pqr cursor for select * from emp;

- cursor is based on SELECT statement.
- you can do partial SELETION as well

declare pqr cursor for select ename, sal from emp;

declare pqr cursor for select * from emp where deptno = 1;

declare pqr cursor for select * from emp order by name;

declare pqr cursor for select dname, ename from emp, dept where emp.deptno = dept.deptno;

cursor based on join

declare pqr cursor for select * from v1;

→ Recommended! ↑ fastr!

create view first & write cursor on view.

cursor based on view:

DATE

∴ Cursors are used for storing multiple rows

- Cursors are used for handling multiple rows
- cursors are used for storing the data temporarily (i.e. RAM)

```
delimiter //
Create procedure abc()
Begin
    declare a int(4);          $ Must match with table columns
    declare b varchar(15);
    declare c int(4);
    declare d int(2);

    declare x int(4) default 1;
    declare c1 cursor for select * from emp;
    open c1;                   <- select is fired, c1 is populated with data
    while x < 6 do
        fetch c1 into a,b,c,d;
        /* processing!
            e.g set hra=c*0.4, etc */

        insert into tempp values (a,b);
        set x = x + 1;
    end while;
    close c1;                  <- RAM occupied by c1 is freed up!
end; //
delimiter ;

call abc();
select * from tempp;
```

cursor declaration → at this pt, cursor doesnot contain data

optional but recommended!

- cursor is the last var. you declare!

x = 1 2 3 4 5 6

a = ~~null~~	b = null	c = ~~null~~	d = null
~~1~~	~~A~~	5000	~~1~~
~~2~~	~~B~~	6000	~~1~~
~~3~~	~~C~~	7000	~~1~~
~~4~~	~~D~~	9000	~~2~~
5	E	8000	2

cursor pointer.

C1

	EMPNO	ENAME	SAL	DEPTNO
~~→~~	1	A	5000	1
~~→~~	2	B	6000	1
~~→~~	3	C	7000	1
~~→~~	4	D	9000	2
→	5	E	8000	2

Fetch will fetch the data from cursor pointer & after fetching the cursor pter is in cremented automatically

O/P

1 A
2 B
3 C
4 D
5 E

classmate

PAGE

- You can not manipulate / update the data in cursor ! (insert, update, delete).

- You can only fetch sequentially (Top to bottom). i.e. cursor pointer moves in only one direction. (it is a singly linked list internally).
  ↳ not a circular singly LL.

→ . cursor is a READ ONLY variable.
- Data that is present in the cursor, it can not be manipulated.
- You will have to fetch 1 row at a time into some variables.
- You can only fetch sequentially (top to bottom)
- You can not FETCH BACKWARDS in a cursor
- You can only FETCH ONE ROW at a time.

- The scope of a cursor is the prog. (Begin to end)
  ∴ It is optional to write close c1; but, it is recommended. It will free up the RAM space. for ~~other~~

- You cannot open the same cursor repeatedly i.e. you will get an error that cursor is already open.
- You will have to close it before you can open it again
- You can declare multiple cursors in a program.

- You can open multiple cursors at a time.

e.g:

```
declare c1 cursor for select * from emp;
declare c2 cursor For select * from dept;

Open c1;
  :
open c2;
  :
  :
```

- Ensure that your server RAM is Sufficient. (High)
- if the cursor size is 10gb & RAMsize is 4GB. it will make use of virtual memory (swap area) in server HD.
  ↳ Slow!

- There is no upper limit on no. of cursors that can be opened at a time.
- How many cursors you want to allow to be open at a time can be controlled by DBA. (∵ Ram is limited).
- init.ora -> it is a startup

x = 1.

if we write.

```
while x < 4 do
    :
```

Will execute 3 times.

O/P.

1 A
2 B
3 C.

if we write. x = 1

```
while x < 11 do
    :
```

O/P.

1 A
2 B
3 C
4 D
5 E

C1

EMPNO	ENAME	SAL	Deptno.
1	A	...	...
2	B	...	...
3	C	...	...
4	D	...	...
5	E	...	...

after fetching 5 rows, it will give an error. (∵ it is a singly non-circular Linked list).

```
delimiter //
create procedure abc()
  begin
    delare a int(4);
    declare b varchar(15);
    declare c int(4);
    declare d int(2);
    declare x int(4) defaut 0;
    declare y int(4);

    declare c1 cursor for select * from emp;
    select count(*) into y from emp;

    oper c1;
      While x<y do
      fetch c1 into a,b,c,d;
        /*
          processing
        */
        insert into tempp values (a,b);
        set x = x+1;
      end while;
    close c1;

end; //
delimiter ;
```

↑ will work w/o giving an ~~error~~.

BUT

it is working because SELECT is simple.

We can not use count(*) with ~~complex~~ complex SELECTs like ~~join &~~ ~~union~~ SELECT with join or union.

DATE

BEST WAY OF WORKING WITH CURSORS

Declare a CONTINUE handler for NOT FOUND

```
delimiter //
  create procedure abc()
    begin
        declare a int(4);
        declare b varchar(15);
        declare c int(4);
        declare d int(2);

        declare finished int(4) default 0;
        declare c1 cursor for select * from emp;

        declare continue handler for not found
finished is still 0 :  ⟶  set finished = 1;
                          not an initialization.
                          ↑ cursor attribute
        ~~select count(*) into g from emp;~~

        open c1;

loop label  cursor_c1_loop : loop
            fetch c1 into a,b,c,d;
            if finished = 1 then
                leave cursor_c1_loop
            end if;

            insert into tempp values (a,b);
        end loop cursor_c1_loop;
      close c1;
  end; //
delimiter ;
```

classmate PAGE

continue handler is an exception handler. & not found is a cursor attribute. on encountering unsuccessful fetch, a 'not found' event is occured. continue handler handles this even & sets finished = 1;

DATE

This program will work for any no. of rows & any kind of SELECT (complex select like based joins, unions etc)

Finished = 0

a = null b = null c = null d = null

EMPNO ENAME

Finished = 0 → 1 A

0 → 2 B

0 → 3 C

0 → 4 D

0 → 5 E

1 →

not found!
∴ set finished = 1

O/P → TEMPP

FIR	SEC
1	A
2	B
3	C
4	D
5	E

'Not found' is a cursor attribute, it returns boolean TRUE if the last fetch was unsuccessful & FALSE value if the last fetch was successful

finished = 0

declare continue handler for not found set finished = 1;

↳ if after every fetched, it is checked if the fetch was success full. If the feteh is successful, it will not execute

If the fetch is unsuccessful, it will execute

DATE

ordinary text file. (you can open & view it)

init.ora → it is a startup file of database.

- This file is automatically executed when you startup the database.
- It is a parameter size.
- this file is read by the RDBMS when you startup the DB.

parameters.
- DB_BLOCK_SIZE = 4096
- DB_CACHE_SIZE = 512000000
- OPEN_CURSORS = 100
- :
- etc. (134 parameters for MySQL & 2352 Oracle)

MAX. no. of open cursors (allowed to be open at a time)
it is controlled by DBA.

→ Value can be changed in the notpad.

But . . .

after changing, you'll need to restart the RDBMS.

∵ the init.ora is read by RDBMS on startup time.

init.ora file parameters are used for performance tuning.

call by ref is always slower than call by value

DATE

Stored Procedures Parameters

Parameters are of three types.

- IN parameters
- OUT parameters
- INOUT parameters

IN parameters (FASTEST in terms of processing speed)

- Read only parameters. You cannot change the value of parameters inside the procedure.
- If you dont want to return a value, use IN Parameter (it will be faster than OUT & INOUT).

OUT Parameters (MOST secure)

- Write only. You can change the value of parameter inside the procedure. but you cant' read. If you want to return value but do not want to read it.

INOUT parameters (BEST FUNCTIONALITY)

- Read and Write
- If you want to return value, use INOUT.

So where to use OUT?

DATE

(Read only)
IN parameters (By default)

∵ it is Read only.

```
delimiter //
create procedure abc ( in y int(4))
   begin          /* set y = 100; */
      insert into tempp values (y, 'inside abc');
   end; //
delimiter ;

call abc (5);          // passing const
select *from tempp;
```

→ O/P

FIR	SEC
5	inside abc

```
set @x = 10;           // passing var
call abc (@x);
select *from tempp;
```

TEMPP

FIR	SEC
10	inside abc

```
set @x = 10;           // passing var
call abc (2 * @x + 5);
```

TEMPP

FIR	SEC
25	inside abc.

DATE

∴ to IN parameters, we can pass a constant, variable, expression.

↳ it is call by value.

By default, parameters are IN parameters.

i.e.

even if we dont use 'IN' with parameters

... abc(y int(4)) ∴ y will still be 'IN' by default.

our parameters (write only) (call by reference)

- Can change the value inside procedure.

```
create procedure abc(out y int(4))
begin
    set y = 100;
end;//
```

↑ Allowed!

```
Set @x = 10;
Select @x from dual;
```

↳ o/p 10

RAM: x = 10 — 40962048

we can ONLY pass VARIABLES to our parameters. (Not consts, expression)

```
call abc(@x);
select @x from dual;
```

RAM: x = 10 100 — 40962048 ← y

Here, since y is out value, the value of @x is not passed. the address is passed. ∴ x & y will share the address.

o/p
100

PAGE

classmate

TRIGGER

STORED FUNCTN

DATE

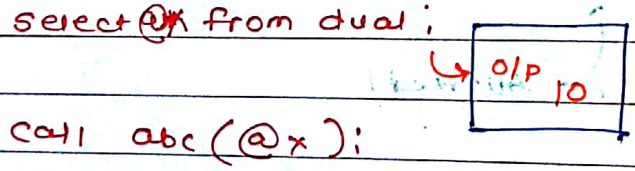
∴ The procedure can 'return' indirectly with OUT parameter. (i.e. We are able to see the changes made by abc() outside abc())

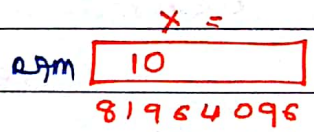
INOUT ~~var~~ parameters (call by reference)
(Read and write)

↳ We can ONLY pass variables

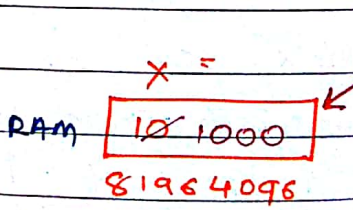
```
create procedure abc( inout y int(4))
    begin
        set y = y* y* y;
    end ;
```

```
set @x = 10;
select @x from dual;
call abc(@x);
select @x from dual;
```





∴ inout is also call by ref, @x & y will share the value '10'.



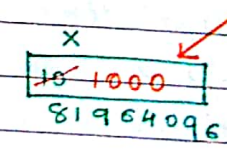
DATE

∴ If you want to return a value (values) from procedure, use OUT or INOUT ~~ref~~ parameters in procedure.

∴ pqr() calls abc()
MySQL doesn't support forward declaration.

CREATE abc() FIRST.

```
delimiter //
Create procedure pqr()
    begin
        declare x int(4) = 10;
        insert into tempp values (x, 'before abc');
        call abc(x);
        insert into tempp values (x, 'after abc');
    end; //
delimiter //

delimiter //
    Create procedure abc(inout y int(4))
    begin
        set y = y*y*y;
    end; //
delimiter ;

call pqr();
select * from tempp;
```



Tempp	
10	before abc
1000	after abc

classmate

PAGE

DATE

∴ Procedures can not return value unless OUT/ INOUT parameters are used. (indirectly returning)
We can not return value directly.
To return value directly, we have FUNCTIONS.

STORED FUNCTIONS

- Functions are the stored objects. i.e. the objects that are stored in the database.
just like (tables, indexes, views, stored procedures)

- Anything we do with CREATE, it is a stored object.

- Stored objects are stored in sys.tables in DB

- Stored function is routine (set of commands) that returns a value.
- function must return something. (compulsory).
i.e you can not have a void function.

↳ Everything else is exactly same as procedure.
just like Stored procedures,
- There's a sys. table for functions also.
- functions are stored in sys table in COMPILED FORMAT!
↳ ↑ faster & hiding the source code.

DATE

- Just like stored procedures, stored functions are also global. (can be accessed from anywhere).
- Function can call procedures & vice versa.
- OVERLOADING OF FUNCTIONS IS NOT ALLOWED!
- With functions, we can only have IN parameters! ∴ theres no call by ref concept here ∴ it is can return value directly.

STORED FUNCTN

- Stored functions are of 2 types.
  - i) Deterministic functions — eg getTime()
  - ii) Not-deterministic functions. — eg getHRA()

For the same input parameters, if the stored function returns the same value, it is considered deterministic. and otherwise the stored function is not deterministic.

You have to decide whether a stored function is determinstic or not.

If you declare it incorrectly, the stored function may produce an unexpected result, or the available optimization is not used which degrades the performance.

TRIGGER

Creating a function

Do not specify width (size)

```
delimiter //
    create function abc()
        returns int
        deterministic
            begin
                return 10;
            end; //
delimiter;
```

Note that we are not using 'call'

```
delimiter //
    create procedure pqr()
        begin
            declare x int(4);
            set x = abc();

            insert into tempp values (x, 'after abc');
        end; //
delimiter ;
```

```
call pqr()
select * from tempp;
```

o/p

TEMPP	
FIR	SEC
10	after abc

We use 'call' only with procedures.

DATE

Except for if statement. (explained later)

Unlike a procedure, a function cannot be called by itself [e.g call abc()] Why ?
because a function returns a value (ALWAYS) and that value has to be stored somewhere.
So, we will have to assign it to a variable or use it in some expression.

```
delimiter //
    create function abc (y int (4))
    returns int
    begin
        return y*y;
    end; //
delimiter ;
```

create it first

```
delimiter //
    create procedure pqr ()
    begin
        declare x int (4);
        set x = abc (10);
        insert into tempp values (x, 'after abc');
    end; //
delimiter ;
```

then create it. (Always create caller later)

```
call pqr();
select * from tempp;
```

TEMPP

FIR	SEC
100	after abc

o/p

DATE

diff & similarities betn functions & procedure?

Similarities

- both are Stored objects
- both are Stored in Sys. table
- both are Stored in compiled ∴ ↑speed
- both are global

Differences

- funn. returns a value.
- function can only allow IN parameters.
- function is call by value.

  procedure is both call by value & reference. (Suppors all parameters IN, OUT, INOUT).

WE CAN call (invoke) a function in SELECT statement.

- also, Stored functions can also be called in SQL commands.

```
create function abc( y int (4))
  returns int
      deterministic
        begin
          return y * y;
      end; //
```

```
select abc(10) from dual;
Select abc(sal) from emp;
delete from emp where abc(sal ): 10000;
```

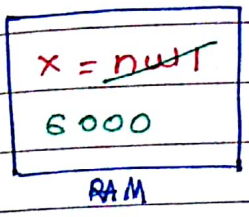
```
delimiter//
  create function abc (y int(4))
     returns boolean
          deterministic
           begin
               if  y>5000 then
                        return TRUE;
              else
                        return FALSE;
              end if;

           end;//
delimiter ;

delimiter //
   create procedure pqr()
   begin
       declare x int(4);

       select sal into x from emp where ename='KING';
       if abc(x) then
            insert into tempp values (x, '>5000');
       else
            insert into temp values (x, '<=5000');
       end if;
   end;//
delimiter ;

call pqr();
Select * from Tempp;
```

DATE

TRIGGER

EMP

ENAME	SAL
KING	6000
:	:



Temp p

FIR	SEC
6000	>5000

- If a function returns boolean, you can call it in IF statement. (W/o assigning it to a variable).
- functions are normally used for validations.

To see which all functions are created:-

Show function Status; <- Shows all the functions in the database

Show function Status where db = 'DAC1';

Show function status where name like 'A%';

To view source code of stored function:-

Show create function abc;

To Share function with other users:-

DAC1 ->MySQL> grant execute on function abc to dac2;

DAC1_MySQL> grant execute on function abc from dac2;

Use schemaname.functionname to call functions of other users.

classmate

PAGE

DATE

Database Triggers (v.imp)

- Triggers are also stored objects. just like tables, indexes, views, procedures, functions.

EMP

ENAME	SAL	DEPTNO
A	5000	1
B	5000	1
C	5000	1
D	3000	2
E	3000	2

department total

DEPTOT

DEPTNO	SALTOT
1	15000
2	6000

output table.

TEMPP

FIR SE

DATE

```
select deptno, sum(sal) from emp group by deptno;
```

slow

DEPTNO	SUM(SAL)
1	15000
2	6000

```
select * from depttot;
```

~~very fast~~

DPTNO	SALTOT
1	15000
2	6000

o/p is same then why does the depttot is created.

group by uses sorting internally. if we have millions of rows, it will be very slow.

∴ Creating a separate table (Denormalization).

Triggers:

- Present in some RDBMS.
- Trigger is a routine (set of commands) that gets executed automatically when some event takes place.
- N to procedure But
  We need to call the procedure for its execution,
  Trigger gets executed automatically on occurrence of some event.

- In front ends, events are like click, double click etc.

- Trigger works on back end events.
- Triggers are written on TABLES.
  Events in mySQL are:- ~~Oracle~~
  Before insert, After insert
  Before Delete, After Delete
  Before Update, After Updak

DATE

```
delimiter //
create trigger abc
before insert
on emp for each row
Begin
    insert into tempp values (1, 'inserted');
end ;//
delimiter ;
```

Trigger name

event

table

* All triggers are at row level. (they will exe. for each & every row)

Before inserting a row in emp table, a row will be inserted in tempp table

Once the trigger is created, it will be executed automatically when we insert into emp table.

```
insert into emp values ('F', 3000, 2);
```

o/p.

one row will be inserted into emp table.

&

Tempp

FIP	SEC
1	inserted.

Uses of triggers:-

- Maintains logs of insersions.
- These logs are called audit trail.
- It is a std practice of maintaining logs of insersions.

First the row is inserted in tempp then row is inserted into emp table.

- All the triggers ar at server level.
that means, you can perform any DML operations using any front-end S/W (from GUI, Forms etc), the triggers will always execute.

- ~~There are~~
- Inside a trigger you can do any processing. (calling procedure, functions etc).

i.e,
- Within the trigger, all MySQL PL commands are allowed.
e.g: variable declaration, if statement, loops, cursors, sub-blocks, etc.
Stored procedure and stored function can also be called from inside the trigger.

- Whenever you rollback or commit, the data will always be consistent.

```
insert into emp
values ('F', 3000, 2);

Rollback;
```

→ Will insert a row in tempp first the a row is inserted in emp;

→ Above inserted rows will ~~not~~ be undone.

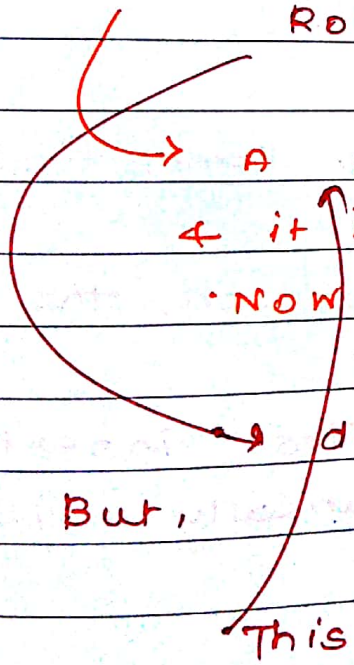
DATE

- Rollback and commit are not allowed inside the triggers. it leads to inconsistency.

eg:

```
create trigger abc
    :
    :
insert into tempp values (1,'inserted');
commit;
end;
    :
```

```
insert into emp values ('F',3000,2);
Rollback;
```



A row will be inserted in TEMPP first & it is committed.

Now row is inserted in emp.

due to rollback, this operation is undone

But,

This operation is not undone.

DATE

- If DML operation on table fails, it will cause the event to fail, then trigger changes are automatically Rolled Back.

- If trigger fails, it will cause the event to fail. then DML operation on your table is automatically Rolled back.

- YOUR DATA WILL ALWAYS BE CONSISTENT.

~~consider~~

e.g insert into emp (....);

~~if for so~~

- a row is inserted to tempp first.

- But, due to some reason, the insert fails.

in this case, the first inserted row. (operation) is automatically rolled back

DATE

Uses:

To maintain logs of insessions (audit trails) often insert trigger is recommended. (will be faster)

```
delimiter //
  Create trigger abc
      After insert
        On emp for each row
        begin
            insert into temp values (1, 'inserted');
        end ;//
delimiter ;
```

DATE

lets assume there's emp2 table with identical structure as Emp

EMP 2

ENAME	SAL	DEPTNO
F	...	...
G	...	...
H	...	...
I	...	...

In MySQL, all triggers are at row level, they will always execute for each and every row.

∴ there are 6 events in MySQL, we can have maximum 6 triggers per table.

- in oracle, we can have statement level triggers and row level triggers.

-In oracle, max 12 triggers per table.

not in mysql.

Statement level trigger -> Fires only once
row level trigger -> fires for each row.

one statement.

```
insert into Emp Select * from emp2;
```

4 insersions.

EMP

ENAME	SAL	Deptno
A	:	:
B	:	:
C		
D		
E		
F		
G		
H		
I		

Temp

1	inserted
1	inserted
1	inserted
1	inserted.

Trigger excecuted 4 times (∵ it ~~was~~ is row level trigger. 'for each row')

```
delimiter //
create trigger abc
    before insert
        on emp for each row
        begin
            insert into tempp values (new.sal,
                                      new.ename)
        end; //
delimiter.
```

Refers to salt ename of currently inserting row.

```
insert int emp values ('F', 3000, 2);
```

o/p:

Temp

FIR	SEC
3000	F

Why to use it?

- To automatically create a copy of every insertion. W/o ~~writing~~.

(Automatic data duplication (Mirroring).

ie. To maintain 2 or more copies of table at the time of insert.

PAGE

DATE

~~It is the~~ This is the concept of parallel servers.
~~data~~ - Data replication ~~can b~~
- Standby database
- Dataguard.

- It is recommended to replicate data on a diff. drive. So if the server crashes we have a backup (shadow table)

- The tempp table in this example is called 'shadow table'.

- To maintain shadow table / standby DB, 'after insert trigger' is recommended. (before insert trigger will be slower)

Data cleansing

```
delimiter //
  create trigger abc
    before insert
      on emp for each row
        begin
          set new.ename = upper(new.ename);
        end;//
delimiter ;
```

if user enters ~~ename~~ ename in lower case but we want it to be in upper.

DATE

insert ('f', 3000, 2);

↳ before inserting this row to emp, the before insert trigger will make 'f' to 'F'.

Auto updation of related data:

```
delimiter //
    create trigger abc
        before insert
            on emp for each row
        begin
            update deptot set saltot = saltot + (2000)
                                        new.sal.
                                        (3000)
                where deptno = new.deptno;
        end;
delimiter ;

insert into emp values ('F', 3000, 2);
```

Emp

ENAME	SAL	DEPTNO
:		
F	3000	2.

DeptTOT

Deptno	saltot
1	15000
2	9000

Auto update on related data.

If you drop the table, then triggers and indexes are dropped automatically.

you can drop trigger by:

drop trigger abc;

show triggers; -> Will show All the triggers.

Show triggers from DAC1 -> .... of DB DAC1.

In Delete trigger:

We can use,

Old.ename, old.sal, old.deptno.

In update trigga, we can use,

old.ename, old.sal, old.deptno,

new.ename, new.sal, new.deptno

DATE

Deleting D & E from ename

```
Delete From emp Where deptno = 2;

delimiter //
Create trigger pqr
    before delete
        On emp for each row
        begin
            insert into tempp values (1,'deleted');
        end;//
delimiter;
```

before deleting from emp, the trigger will first insert to temp.

Uses:

- To maintain logs of deletions (audit trails)
- To maintain such logs, after delete trigger is recommended.

Tempp

FIR	SEC
1	deleted
1	deleted.

DATE

```
delimiter //
  create trigger par
      before delete
        on emp for each row
          begin
            insert into tempp values(old.sal, old.ename);
          end;//
delimiter ;
```

```
delete from emp
        where deptno=?;
```

old is a syst. temporary table. before deleting a row, from emp

Tempp

Fir	sec.
3000	D
3000	E

Known as History table.

In real life, we never delete a row, we store it somewhere else.

DATE

Temporary Sys-table made by MySQL.

OLD

ENome	SAL	DEPTNO
D	3000	2
E	3000	2

Before a row is deleted from 'emp' table, the row is inserted to a system table named 'OLD'.

```
delimiter //
    create trigger pqr
        before delete
            on emp for each row
            begin
            update deptot set saltot = saltot -
                                    old.sal
                where deptno = old.deptno;
            end; //
delimiter;
```

EMP

ENAME	...	...
A	...	...
B	...	...
C	...	...

DEPTOT

DeptNo	SALTOT
1	15000
2	0

Item

update triggers

```
update emp set sal = 6000
        where deptno = 1;
```

```
delimiter //
  create trigger xyz
     before update
        on emp for each row
          begin
            insert into temppvalues (1, 'updated');
          end;//
delimiter ;
```

before ~~the~~ ~~row~~ ~~each~~ row ~~is updated~~, ~~each~~ (3 ~~rows~~) updating a rows in emp table

a row is inserted

Tempp

FIR	BEC
1	updated
1	updated
1	updated

- To maintain updation logs.
- After update trigger is recomended for update logs

DATE

Cascading triggers.

. One trigger causes second trigger to execute, which in turn causes 3rd trigger to execute and so on.

in prev. eg. in trigger xyz, there is a insert stmt on tempp table.

What if Tempp has an insert trigger is set on Tempp table?

↳ it will execute that too!

⟹ Cascading triggers.

- In MySQL, and Oracle, there's no upper limit on the no.

- MS SQL server, max 32 levels of cascading triggers.

∴ MySQL & Oracle RDBMS are used for complex enterprise applications.

If some cascading triggers causes one of the previous triggers to execute, it will not go into infinite loop, you will get an error of mutating tables & the entire transaction is automatically rolled back.

DATE

In case of power failure, N/w failure, reboot, window close, improper exit, system failure, the entire xaction is automatically rolled back.

Inside update trigger, we can use both 'new' & 'old'

```
delimiter //
create trigger xyz
    before update
        on emp for each row
            begin
                insert into tempp values (old.sal,
                                          old.ename);
                insert into tempp values (new.sal, new.
                                          ename);
            end;//
delimiter ;
```

update emp set sal = 6000 where deptno=1

Tempp

FIR	SEC	
5000	A	← old values of ROW
6000	A	← updated values.
5000	B	
6000	B	
5000	C	
6000	C	

SHADOW & HISTORY Table

↳ after update is recommended.

classmate

PAGE

DATE

```
update EMP
    set SAL = 6000
        where ENAME = 'A';

delimiter //
  create trigger xyz
      before update
          on emp for each row
              begin
                  update deptot set saltot = saltot -
                                  old.sal + new.sal
                              where deptno = old.sal ;
              end ; //
delimiter ;
```

(15000) (5000) (1) (1)

DEPTOT

deptot	saltot
1	16000
2	6000

```
update emp
    Set ename = 'AMIT'
        Where ename = 'A';
```

only valid for update trigger.

```
create trigges xyz
    before update of sal
        on emp for each row
            begin
                update deptot set Saltot = Saltot - old.sal
                        + new.sal
                            Where deptno = old.deptno;
                end;//
    delimiter;
```

'before update of sal ->
Only if the SAL is updated the trigges will execute.
If ENAME or any other column is updated the trigger will not execute (to prevent the unnecessary processing)

DATE

```
update  EMP
        set DEPTNO = 2
        where ename = 'A';
```

This trigger will execute if either of them is updating or both are updating.

```
delimiter //
create trigger xyz
before update of sal, deptno
on emp for each row
begin
    if updating ('DEPTNO') then
        update deptot set saltot = saltot - ol.sal
        where deptno = old.deptno;
        update deptot set saltot = saltot + new.sal
        where deptno = new.deptno;
    else
        update deptot set saltot = saltot - old.sal +
                                   new.sal
        where deptno = old.deptno;
    end if;
end; //
delimiter;
```

This is case sensitive ∴ be careful

function returns boolean.

deptot

Deptno	saltot
1	10000
2	11000

DATE

you can't combine before & after.

```
delimiter //
create trigger xyz
    before insert or delete or update of sal, deptno
        on emp for each row
        begin
            if inserting(...) then
                :
                :
            elseif deleting (...) then
                :
                :
            elseif updating('DEPTNO') then
                :
                :
            else
                /* if you update only the
                   SAL column */
                :
            end if;
    end; //
delimiter ;
```