Data types in mysql

- Numeric
- Date and Time
- String Types.

Let us now discuss them in detail.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions –

- **INT** – A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** – A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** – A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** – A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE(M,D)** – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** – An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M)

Date and Time Types

The MySQL date and time datatypes are as follows –

- **DATE** – A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** – A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** – A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** – Stores the time in a HH:MM:SS format.
- **YEAR(M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

String Types

Although the numeric and date types are fun, most data you'll store will be in a string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
- **Nvarchar** stores Unicode or Non-English character data types, and it can contain a maximum of 4000 characters. It supports ASCII values as well as special characters. To support multiple languages, nvarchar is a must.

- **BLOB or TEXT** – A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBs and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.
- **TINYBLOB or TINYTEXT** – A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** – A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** – A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

- **ENUM** – An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

constraints in the table

These are of 2 types

1. field level constraint- the constraint which you need to write immediately after column declaration are column level constraints
2. Table level constraint--- the constraints which are based on multiple columns, or dependent on some other column, then it is good to use table level constraint.

constraints

Not null	field level	It does not allow to add null values in the table
default	field level	If user does not specify any value, then instead null what value can be added in the column is defined with default constraint salary decimal(9,2) default 10000
primary key	table level	It does not allow null values and duplicate values, so values should be unique and not null In a table there can be only one primary key.
unique	field level	It does not allow duplicate values, but allows any number of null values
check constraint	table level	Before entering data in the column, if you want to

		check some condition, and if the condition is true, then only store the data, otherwise show error
foreign key constraint on delete <action> on update<action>	table level	while entering value in the column, if you want to refer the value of primary key of the same table or different table

```
create table myemp(
empid int primary key,
ename varchar(30) not null,
sal float(9,2) check(sal>0),
desg varchar(10) default 'Analyst');
```

To delete the existing table

```
drop table myemp;
```

```
create table student(
   sid int primary key,
sname varchar(30),
address varchar(20)
)
```

```
create table course(
   cname varchar(20) primary key,
  duration int check(duration >0),
description varchar(20)

)
```

```
create table stud_marks(
studid int,
cname varchar(20),
marks int check(marks between 0 and 100),
primary key(studid,cname),
constraint fk_nm foreign key(studid)
references student(sid)
on delete cascade
on update cascade,
constraint fk_cnm foreign key(cname)
references course(cname)
on delete cascade
on update cascade
```

	)

create table category(

cid int primary key,

cname varchar(20),

description varchar(30))

create table product(

pid int primary key,

pname varchar(20),

qty int,

catid int,

constraint fk_cid foreign key(catid) references category(cid)

on delete set null

on update cascade)

on delete (action)

on update (action)

action values.

cascade	if delete from parent table, then delete matching rows from child table, if update from parent table, then change the value is child table also
set null	if delete from parent table, then set value to null in matching rows from child table, if update from parent table, then set value to null in matching rows in child table
no action	if on delete and on update clause is not used, then values from the parent table can be deleted only if, the corresponding value does not exists in the child table

DML statements

insert	insert into <table> values(<values for all columns>)	insert into mytable values(12,'xxx','pune')

	insert into<table>(list of columns) values(<values for specified columns>)	insert into mytable(id,name) values(13,'yyy')
update	update <table> set fiedl1=val1,field2=val2, ... where <condition> if where cluse is not used then it will update all the rows	update mytable set name='ccc',addr='yyy' where id=12
delete	delete from <table> where <condition> if where cluse is not used then it will delete all the rows	delete from mytable where id=12
to insert multiple rows in a table	INSERT INTO tbl_name (a,b,c) VALUES (1,2,3), (4,5,6), (7,8,9);	insert into salesman(sid,sname,address) values(11,'xxx','Pune'), (12,'yyy','pune'), (13,'zzzz','mumbai');

to delete the table ----drop table will delete all the rows and will also delete table from the databse

drop table <table name>

drop table emp;

to delete all the rows from the table

truncate table <table name>---it will delete all the rows from the table, but keep empty table as it is.

To set autocommit off

set autocommit=0;

To set autocommit on

set autocommit=1;

Trucate table

truncate table product

delete from product;

Tucate	delete
it is a DDL, so it is autocommit	it is dml statement, changes need to commited
rollback is not possible	rollback is possible

we can not use where cluse	where clause can be used

alter table

To change the table structure use Alter table <table name>

add column	alter table mytable add addr int first|after id;	alter table mytable add addr11 varchar(20) after id alter table mytable -> add addr11 varchar(20) after id, -> add addr12 int first;
delete the column	alter table <table name> drop column <colname>	alter table mytable drop column addr11
modify the existing column	alter table <table name> modify <existing column name> <modified details>	alter table mytable modify location varchar(50) alter table mytable modify location varchar(50) not NULL
rename the column	alter table <table name> CHANGE COLUMN old_name new_name column_definition	alter table mytable change location mylocation varchar(50)
add constraint	alter table < table name > add constraint <cn> foreign key ..... alter table < table name > add primary key(id)	
delete constraint	alter table < table name > drop constraint <cn>	
rename table	alter table <old table name> rename to <new table name>	alter table mytable rename to mytable11

ALTER TABLE User

ADD CONSTRAINT userProperties

FOREIGN KEY(properties)

REFERENCES Properties(ID)

create owner table to store (ownerid, oname,address)

create vehicle table to store(vid,vname, model,company,ownerid,sid)

ownerid is a foreign key references owner table (ownwerid)

sid is a foreign key references salesman table (sid)

create a table salesman to store (sid, sname, address)

<table>
<tr><td>create table owner(
ownerid int primary key ,
oname varchar(20),
oaddress varchar(20))</td><td rowspan="2">create table vehicle(
vid int primary key ,
vname varchar(20),
model varchar(20),
company varchar(20),
ownerid int ,
sid int ,
constraint f_ownerid foreign key (ownerid)
references owner(ownerid)
on delete set null
on update cascade ,
constraint f_sid foreign key (sid) references
salesman(sid)
on delete set null
on update cascade);</td></tr>
<tr><td>create table salesman(
sid int primary key,
sname varchar(20),
saddrerss varchar(20))</td></tr>
</table>

1. to list all employees from dept 10
   ```
   select deptno,empno,ename
   from emp
   where deptno=10;
   ```
2. to list all employees who are working in smith's department

   ```
   select *
   from emp
   where deptno=( select deptno
                  from emp
                  where ename='smith'
                  );
   ```

nested queries

1. simple nested query
   when output of one query is dependent on output of another query, then we use nested query,
   outer query is called as parent query, and inner query is called s child query
   if the child query is not dependent on parent query data, then it is called as simple query.
2. co related query

when output of one query is dependent on output of another query, then we use nested query,

outer query is called as parent query, and inner query is called s child query

if the child query is dependent on parent query data, then it is called as co-related query.

examples

1. to list all employees with sal < avg(sal) of departmrnt 10?

```
select * from emp
where sal < (select avg(sal)
            from  emp
            where deptno=10)
```

2. display all employees who are working in ward's department

```
select * from emp
where deptno=(select deptno
                  from emp
            where ename='ward')
```

3. to list all employees with sal < avg sal of wards department

```
select *
from emp
where sal <(select avg(sal)
            from emp
            where deptno=(select deptno
                          from emp
                          where ename='ward'))
```

4. to display all employees with sal > smith's sal and jones sal

```
select *
from emp
where sal > all(select sal
            from emp
            where ename in('smith','jones'))
```

5. to display all employees with sal > either smith's sal or jones sal

```
select *
from emp
where sal > any(select sal
            from emp
            where ename in('smith','jones'))
```

6. to display all employees who are working either smith's dept or jones dept

```
select *
from emp
where deptno in (select deptno
                 from emp
                   where ename in('smith','jones'))
```

7. to find all employees with sal < avg sal of its own department

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

8. to find all employees whose sal < its own mgr sal

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

9. list all employees who are working on accounting department

```
select *
from emp e
where deptno=(select deptno
              from dept
              where dname='Accounting')
```

10. display all departments in which no employees are there

```
select *
from dept d
where   not exists (select *
      from emp e
where e.deptno=d.deptno)
```

11. display all employees who are not mgr of any employee

```
select *
from emp e
where not exists (select *
                  from emp m
                 where m.mgr=e.empno)
```

12. list all faculties who are not assigned to any course

```
select * from faculty f
where not exists (select * from course c where f.fid=c.fid)
```

13. list all faculties who are assigned to some course

```
select * from faculty f
where exists (select * from course c where f.fid=c.fid)
```

14. list all rooms which are not assigned to any course.

```
select * from room r
where not exists (select * from course c where r.rid=c.rid)
```

15. list all courses for which no room is assigned

```
select *
from course
where rid is null
```

16. update sal of smith to jones sal+2000

```
update emp
set sal=(select sal from emp where ename='jones')+2000
where ename='smith'
```

17. delete all records who are working in either smiths dept or jones dept

```
delete from emp
where deptno in (select deptno
                 from (select * from emp ) e
                  where e.ename in ('smith','jones'))
```

18. create table emp_10

```
as
(select * from emp
where deptno=10)
```

19. create table emp_10

```
as
(select * from emp
where 1=2)
```

20. find all employees with sal > smith's sal and sal < ward's sal

```
select * from emp
where sal between (select sal from emp where ename='smith') +1and (select sal
from emp where ename='ward')-1
```

21 . list all employees who are working in jones dept and sal >2000

```
select * from emp

where deptno=(select deptno from emp where ename='jones') and sal>2000
```