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.

Types of keys

primary key	minimal set of columns which identifies the row uniquely is called as primary key There will be only one primary key if primary key contains single column then it is called as primary key if primary contains more than one column, then it is called as composite primary key
candidate key	all possible minimal set of columns which identifies the row uniquely is called as candidate key
surrogate key	if primary key has multiple columns and if managing the data is difficult, then we add one extra column to make the row unique, is called as surrogate key
alternate key	all candidate keys which are not chosen as primary key is are called as alternate key
super key	any combination of columns which identifies the row uniquely is called as super key
foreign key	it is a column which refers column of primary key or unique column of other table or of same table, then it is called as foreign In a table there can be more than one foreign key

examples

1. in room booking table
   (roomno,rname,date of booking, custid, price)
   primary key---roomno+date of booking
   candidate key--- roomno+date of booking
   alternate key---no alternate key
   super key ---→ roomno+date of booking , roomno+date of booking+rname, roomno+date of booking+price, roomno+date of booking+custid,+ roomno+date of booking+custid+price
   foreign key-----roomno of booking table is a foreign key, it should refer roomno in room table
   custid in booking table is a foreign key, it should refer customer table's customerno

2. doctor patient appointment system
   (drid, pid, date,time,amt,prescription)

primary key-→pid+date+time
candidate keys-→ pid+date+time, drid+date+time
alternate key-→ drid
foreign key --→ drid is foreign key, refers to doctor table drid
pid is foreign key, refers to patient table ptid

constraints on the table

<table>
<tr><td>primary key</td><td>it does not allow duplicate values and null values
It is table level constraints, there will be only one primary key in the table</td></tr>
<tr><td>unique key</td><td>it allows only unique values if it is not null values, but many null values are allowed in the column
It is table level constraint</td></tr>
<tr><td>check(condition)</td><td>while storing value in a column if you want to check some condition, if condition is true, then only insert the value in the table
It is table level constraint</td></tr>
<tr><td>not null</td><td>null values are not allowed,
it is field level constraint</td></tr>
<tr><td>default</td><td>when user do not enter the value then default value gets added, instead of null</td></tr>
<tr><td>foreign key</td><td>to maintain correctness of data, if the column refers some other column of different or same table
with this constraint we can use 2 more constraints
on delete <action>
on update <action>

values of action can be
<table>
<tr><td>set null</td><td>changes done in parent table, then corresponding values should set to null in child table</td></tr>
<tr><td>cascade</td><td>the change made in parent table, the same change should be done in child table</td></tr>
<tr><td>no action</td><td>if we donot add on delete and on update statements, the it is no action by default</td></tr>
</table>
</td></tr>
</table>

create student table to store student id, name,mobile, and age
name cannot be blank, mobile cannot be blank, and no duplicate values
age should be in range 20 to 100

create course table to store course id, cname, duration of the course capicty of the course, course capacity should be always < 300

create stud_course table to store
sid, cid marks

<table>
<tr><td>create table student(
sid int primary key,
sname varchar(20) not null,
mobile int unique not null,
age int check(age between 20 and 100))</td><td rowspan="2">create table stud_course(
stuid int,
coursid int,
marks int check(marks between 0 and 100),
primary key(stuid,coursid),
constraint fk_sid foreign key(stuid)
references student(sid)
on delete cascade
on update cascade,
constraint fk_cid foreign key(coursid)
references course(cid)
on delete cascade
on update cascade)</td></tr>
<tr><td>create table course(
cid int primary key,
cname varchar(20) unique not null,
duration int,
capacity int check(capacity <=300))</td></tr>
</table>

to drop the table
drop table <tablename>
drop table student;

create table mytable(
id int primary key auto_increment,
name varchar(20));

insert into mytable values(default,'xxxx')
insert into mytable(name) values('xxxx')

to modify start value
alter table mytable auto_increment=1000;

auto_increment ---- will generate unique values automatically, it can be used only with int columns, and on primary key which has only single column.

DML statements

all DML statements work on single table

to insert record	insert into <table name> values(<list of values>) insert into <table name>(<list of columns>) values(<list of values>)
to delete record if we skip where clause then it will delete all records but the empty table will remain	delete from <table name> where <condition>
to update data if we skip where clause then it will update all records	update <table name> set col1=val1,col2=val2,col3=val3 where <condition>

Truncate table

truncate table mytable;

it will delete all records but will not delete the table, so empty table will remain

Truncate Vs Delete

truncate is DDL statement, DDL statements are autocommited	Delete is DML statement
we cannot rollback the changes	changes can be roll back
where clause cannot be used	where clause can be used

Alter table

To modify table structure we use alter table

add column	alter table mytable add addr varchar(20) not null after id;
delete a column	alter table mytable drop addr;
modify column definition	alter table mytable modify name varchar(50)
modify column name	alter table mytable change name newname varchar(20)
add constraint	1. add a primary key constraint alter table mytable add primary key(id) 2. add foreign key alter table mytable add constraint fk_1 foreign key(location) references mylocation(lid)
drop constraint	alter table mytable drop primary key alter table drop constraint fk_1
rename the table	alter table mytable rename to mytable1;
can be used in auto_increament	alter table mytable auto_increment=1000;

Nested query

1. simple nested query--- in these types of queries, child query gets executed only once and then outer query will get executed
2. co-related query--- in these types of queries, child query gets executed once for each row in the outer query table.

If the child query is not dependent on outer query, then it is called as simple nested query

but if child query is dependent on outer query, then it is called as co-related query.

1. find all employees who are working in smith's or allen department
   select * from emp
   where deptno in (select deptno

   from emp

   where ename in ('SMITH','allen')
2. find all employees whose sal >both smith's and allen department

select * from emp

where sal > any (select sal

from emp

where ename in ('SMITH','allen')

3. Write all queries to find all employee who are working under allen's mgr

   select * from emp
   where mgr=(select mgr from emp
   where ename='Allen')

4. find all employees with sal < avg salary of accounting department.
   select * from emp
   where sal<(select avg(sal) from emp where deptno =(select deptno
   from dept
   where dname='accounting'));

5. find all employees with sal > average sal of its own department.
   select *
   from emp e
   where e.sal >(select avg(sal) from emp d where d.deptno=e.deptno)

   in corelated query usually we use 2 operators
   1. exists--→ it will return true if child query returns one or more rows in the o/p, false otherwise
   2. not exists---→ it will return true if child query does not return any rows in the o/p, false otherwise.