Table of Contents

1. Databases in MySQL

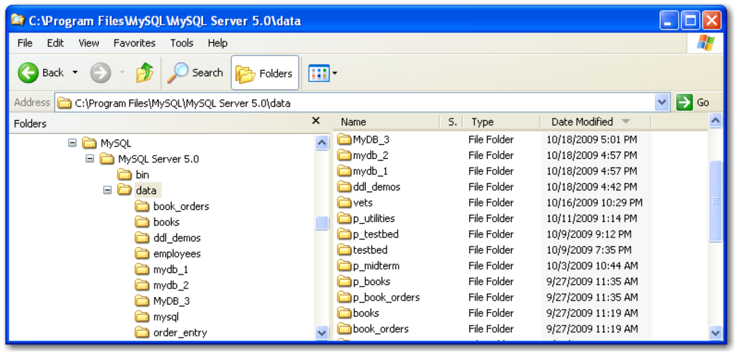
You can create a database in MySQL with either of the following SQL statements:

```
Create Database MyDB_1;
Create Schema   MyDB_2;
```

When you use Create Database in MySQL you get what is commonly called a schema. MySQL uses create schema as an alias for create database. You can create relationships across schema.

In MySQL a database is implemented as a directory in the file system. The tables are implemented as files in that folder.

The default location for the directory for a database varies with your operating system and the version of MySQL. With version 5.0 on a Windows machine, the default directory is a subdirectory of the directory where MySQL stores the dbms program files.

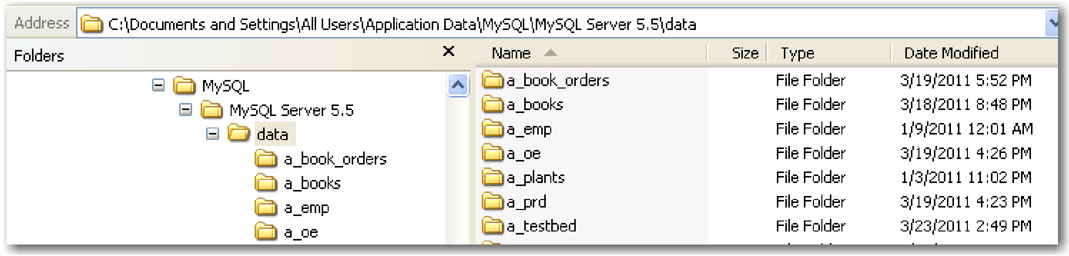


C:\Program Files\MySQL\MySQL Server 5.0\data

```
MySQL
  MySQL Server 5.0
    bin
    data
      book_orders
      books
      ddl_demos
      employees
      mydb_1
      mydb_2
      MyDB_3
      mysql
      order_entry
```

Name	S.	Type	Date Modified
MyDB_3		File Folder	10/18/2009 5:01 PM
mydb_2		File Folder	10/18/2009 4:57 PM
mydb_1		File Folder	10/18/2009 4:57 PM
ddl_demos		File Folder	10/18/2009 4:42 PM
vets		File Folder	10/16/2009 10:29 PM
p_utilities		File Folder	10/11/2009 1:14 PM
p_testbed		File Folder	10/9/2009 9:12 PM
testbed		File Folder	10/9/2009 7:35 PM
p_midterm		File Folder	10/3/2009 10:44 AM
p_books		File Folder	9/27/2009 11:35 AM
p_book_orders		File Folder	9/27/2009 11:35 AM
books		File Folder	9/27/2009 11:19 AM
book_orders		File Folder	9/27/2009 11:19 AM

With version 5.5 on a Windows machine, the default directory is a subdirectory of the Documents and Setting directory.



Address C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server 5.5\data

Folders
- MySQL
  - MySQL Server 5.5
    - data
      - a_book_orders
      - a_books
      - a_emp
      - a_oe

Name	Size	Type	Date Modified
a_book_orders		File Folder	3/19/2011 5:52 PM
a_books		File Folder	3/18/2011 8:48 PM
a_emp		File Folder	1/9/2011 12:01 AM
a_oe		File Folder	3/19/2011 4:26 PM
a_plants		File Folder	1/3/2011 11:02 PM
a_prd		File Folder	3/19/2011 4:23 PM
a_testbed		File Folder	3/23/2011 2:49 PM

You could also create a folder in that directory and it will appear as a database when you give the show databases command.

This is part of the result of the show databases command given by a user logged in as root. An account cannot see a database unless (1) the account has root privileges or (2) the account has permission to access that database.

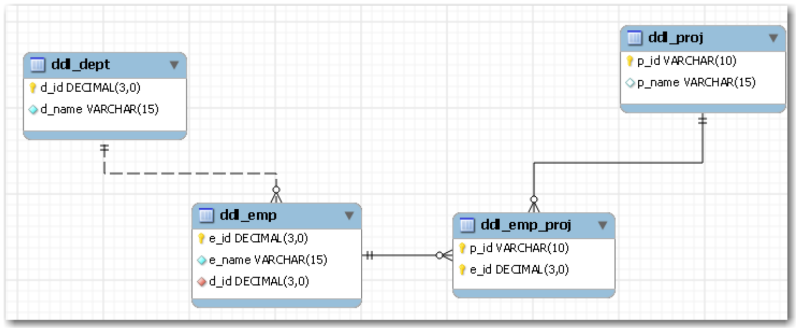
```
mysql> show databases;
+---------------------+
| Database            |
+---------------------+
| information_schema |
| a_a_created_folder |
| a_book_orders       |
| a_books             |
| a_emp               |
| a_oe                |
```

For Window users, the folder and filenames are not case specific and any difference in case does not make a difference- but on a Unix based system the case of the folder/directory and file name matters. If you name the folder with upper case letters and try to use it with lower case letters in MySQL, you will get an error. You should have a consistent naming pattern and MySQL recommends lower case letters.

Another issue with MySQL is that it supports multiple storage engines; the only engine we are using in this class is the InnoDB engine.

2. SQL to create tables

These queries provide a way to use SQL to create relations, change their design, set relationships, create and drop indexes. These statements are considered Data Definition Language statements. This section shows the queries need to create the relations and relationships shown below.



Some of the demo code will create tables that are used to illustrate a technique. You can delete those tables later. These table names will all start ddl_. I will be creating the tables in the a_testbed database.

3. Removing a Table

You can drop a table and all of its data with the Drop Table query. The DBMS does not ask you if you are sure that you want to drop the table.

Demo 01: Dropping a table

```
   DROP TABLE ddl_emp_proj;
```

```
Table dropped.
```

If you try to drop a table that does not exist you will get an error message. In MySQL use the phrase `if exists` to avoid the error message. So you can have drop table commands at the start of a script that creates tables. It is your responsibility to be certain that you do not drop the wrong tables in error.

```
   DROP TABLE if exists ddl_emp_proj ;
   DROP TABLE if exists ddl_proj ;
   DROP TABLE if exists ddl_emp ;
   DROP TABLE if exists ddl_dept ;
```

One time you may find that you cannot stop a table is if the table is involved in a relationship as the parent. You cannot drop the parent table if there are rows in the child table. The above set of drops starts with the child table and then works up to the parent tables.

4. Simple Table Creation

The minimal SQL statement to create a table requires that you give the table a name and that you give each column a name and a data type. The table has to have at least one column defined. This statement does not set a primary key, default value, foreign key references, or other rules about the data- commonly called constraints. All this SQL statement does is create the table structure.

You cannot have two tables with the same name, so if you are experimenting with these statements you might need to drop one or more of these tables. MySQL also enforces rules about dropping tables that are involved in relationships.

For this class all of our tables use the InnoDB engine. On many systems this is the default database engine; if you omit the engine= clause; you get the default engine.

Demo 02: Minimal statement to create a table with two columns

```
   CREATE TABLE ddl_dept2
   (
      d_id      numeric(3)
    , d_name    varchar(15)
   ) engine=InnoDB;
```

```
Table created.
```

When you declare a data type for a column, you limit the values that can be entered in that column. In the above table, the values for d_name cannot exceed 15 characters. MySQL will not truncate string values to fit the defined data type precisions. The values for d_id values must be in the range -999 to +999 due to the numeric(3) designation. MySQL will round decimal values to fit the precisions defined for a numeric type; the value 2.87 will be rounded to3 for the d_id attribute but the value 2345 will be rejected..

4.1. Add a Single New Row Using a Value List

In order to understand table creation you need to test with insert statements. This is a model to insert a single row into the table specifying the column names. This is the preferred technique.

```
insert into my_table (col1_name, col2_name,  . . )
            values   (value1_name, value2_name,  . .);
```

We will look at variations of the insert statement next week.

These are examples of Insert statements that may work- or not

Demo 03: Insert statement that works

```
insert into ddl_dept2 (d_id, d_name)
values(10,'Sales');
```

Demo 04: Insert statement that works; the default is that the columns can accept nulls

```
insert into ddl_dept2 (d_id, d_name)
values(null,null);
```

Demo 05: Insert statement that works; we did not set a primary key so we can have two rows with the same value for d_id.

```
insert into ddl_dept2 (d_id, d_name)
values(10,'Marketing');
```

Demo 06: Insert statement that works; the value for d_id is rounded to 41. This is an implied cast.

```
Insert into ddl_dept2 (d_id, d_name)
values(40.8,'Development');
```

Demo 07: Insert statement that works; We did not restrict negative numerics

```
insert into ddl_dept2 (d_id, d_name)
values(-44,'');
```

Demo 08: Insert statement that fails; value for d_id is larger than specified precision allowed for this column (Out of range value adjusted for column 'd_id')

```
insert into ddl_dept2 (d_id, d_name)
values(1234,'Research');
```

Demo 09: Insert statement that fails; value for d_name is too large for column

```
Insert into ddl_dept2 (d_id, d_name)
values(20,'Accounting and Payroll');
```

Demo 10: Insert statement that fails; not enough values; we specified two columns in the list but provided only one value. Note that this does not use a null for the missing value-

```
insert into ddl_dept2 (d_id, d_name)
values(35);
```

Demo 11: Insert statement that works; this specifies one column in the list and we provide one value. In this case d_name gets a null entry.

```
insert into ddl_dept2 (d_id)
values(35);


Select * from ddl_dept2 order by d_id ;
+------+-------------+
| d_id | d_name      |
+------+-------------+
| NULL | NULL        |
|  -44 |             |
|   10 | Sales       |
|   10 | Marketing   |
|   35 | NULL        |
|   41 | Development |
+------+-------------+

Drop table ddl_dept2;
```

5. Creating Tables with Integrity

You want your database to have integrity- you want to ensure that as far as possible the values in your data are correct. There are three types of integrity:

- Entity integrity- each row should be uniquely identifiable- enforced by primary key constraints
- Referential integrity- a row in a child table needs to be related to a row in a parent table- enforced by foreign key constraints.
- Domain integrity- a data value meets certain criteria- in its simplest form enforced by the data type; this can be more closed defined by a check constraint.

To create a usable database you would want to set these constraints— rules about what data values can be entered in the tables and how the tables are related to each other. Each constraint is an object and has a name. MySQL will create constraint names for you if you do not create your own names. You will see many table creation statements that do not create constraint names and that let MySQL create the names. This approach will make it harder to manipulate the constraint later. Constraint names should reflect the table name, the attribute name, and the type of constraint. The pattern I follow of ending constraint names with tags such as _pk, fk etc is common but is not a syntax requirement. Constraint names must be unique within the schema.

A constraint can be declared on the same line as the column declaration (a column constraint); the constraint may be declared in the create statement after all of the columns are declared (a table constraint). You can also add constraints to an existing table.

At a minimum, when you create a table, you would set the primary key and designate which attributes cannot be left empty.

Constraints that you create when you create the table, or that you add to the table, should be enforced by the DBMS. These are called declarative constraints. (MySQL does not enforce all constraints that you can include in your SQL.) Other rules about data can be enforced with triggers- these are called procedural integrity constraints. We are not discussing triggers in this class. (see CS155P for triggers.)

5.1. Primary key, not null and unique constraints

Demo 12: Creates the project table, sets the pk. Chose a constraint name that reflects its purpose. Do the named constraints as table constraints.

```
CREATE TABLE  ddl_proj
(
   p_id   varchar(10)
 , p_name varchar(15)    null
 , constraint  ddl_proj_PK  primary key(p_id)
 , constraint ddl_proj_P_Name_UN   unique(p_name)
) engine=InnoDB;
```

Demo 13: Creates the department table. Sets additional constraints. If you created this table earlier be sure to drop the previous version.

```
CREATE TABLE  ddl_dept
(
    d_id     numeric(3)
 ,  d_name   varchar(15) not null
 ,  constraint ddl_dept_PK         primary key(d_id)
 ,  constraint ddl_dept_D_Name_UN   unique(d_name)
) engine=InnoDB;
```

Primary Key

Setting a primary key constraint also means that the attribute cannot be null and must be unique with the table.

You can declare a primary key constraint on sets of columns using a table constraint. In that case use the syntax shown here where col1 and col2 are already defined as columns in the table.

```
constraint constraint_name_PK   primary key (col1, col2)
```

Null, Not Null

The default setting for nullability in most dbms is that the column can accept null values. Since defaults can be changed, it is a good idea to always specify Null or Not Null for attributes. The NOT NULL constraint is often not given a constraint name. This constraint can be set only as a column constraint.

Unique

The unique constraint means that no two rows in the table can have the same value for that attribute . Setting a column to be Not Null and Unique means that it is an identifier and is a candidate key for this table. You can have only one primary key defined per table; you can define as many candidate keys as you need for a table.

When you define an attribute to be null and unique, MySQL will let you have multiple rows in the table with a null value for that column. This agrees with the attitude that a null does not equal another null.

You can declare a unique constraint on sets of columns using a table constraint. In that case use the syntax shown here where col1 and col2 are already defined as columns in the table.

```
constraint constraint_name_UN   unique(col1, col2)
```

Demo 14: The Unique constraint on a two column set

```
create table ddl_un (id numeric(3)
, city varchar(15), state char(2)
, constraint location_UN  unique(city, state)
) engine=InnoDB;
```

The first three inserts are ok

```
Insert into ddl_un (id, city, state) values (1, 'Chicago', 'IL');
Insert into ddl_un (id, city, state) values (2, 'Chicago', 'CA');
Insert into ddl_un (id, city, state) values (3, 'Pekin', 'IL');
```

This one fails because it is a duplicate of (city, state)

```
Insert into ddl_un (id, city, state) values (4, 'Pekin', 'IL');
```

```
ERROR 1062 (23000): Duplicate entry 'Pekin-IL' for key 1
```

What about nulls? These are both allowed; one null is not equal to another null. (note this is not true for all dbms)

```
Insert into ddl_un (id, city, state) values (5, null, null);
Insert into ddl_un (id, city, state) values (6, null, null);


select * from ddl_un;
+------+---------+-------+
| id   | city    | state |
+------+---------+-------+
|    1 | Chicago | IL    |
|    2 | Chicago | CA    |
|    3 | Pekin   | IL    |
|    5 | NULL    | NULL  |
|    6 | NULL    | NULL  |
+------+---------+-------+

drop table ddl_un;
```

5.2. Foreign key constraints

The department table and the project table are parent tables and can be created independently. The Employee table is a child of the Department table; therefore you have to make the Department table before the Employee table. The referenced attribute in the parent table must have been defined as the PK or with a Unique constraint. This is so that a child row refers to a unique parent row.

Demo 15: Creates the employee table and creates the relationship object to the department table.

```
CREATE TABLE  ddl_emp
(
  e_id    numeric(3)
, e_name  varchar(15) not null
, d_id    numeric(3)  not null
, constraint ddl_emp_pk  primary key(e_id)
, constraint ddl_emp_dept_fk foreign key (d_id) references ddl_dept(d_id)
) engine=InnoDB;
```

Demo 16: MySQL has a command show create table which will be helpful with some of the MySQL issues

```
show create table ddl_emp\G
*************************** 1. row ***************************
       Table: ddl_emp
Create Table: CREATE TABLE `ddl_emp` (
  `e_id` decimal(3,0) NOT NULL DEFAULT '0',
  `e_name` varchar(15) NOT NULL,
  `d_id` decimal(3,0) NOT NULL,
  PRIMARY KEY (`e_id`),
  KEY `ddl_emp_dept_fk` (`d_id`),
  CONSTRAINT `ddl_emp_dept_fk` FOREIGN KEY (`d_id`) REFERENCES `ddl_dept`
(`d_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- The names of the table and columns are shown enclosed in back ticks (`)
- The primary key constraint is created
- The foreign key constraint is created

Demo 17: Creates the employee project table with a composite primary key. The composite pk has to be defined on a separate line as a table constraint. This also creates the relationships to the employee and project tables.

```
CREATE TABLE  ddl_emp_proj
(
  p_id  varchar(10) not null
, constraint ddl_emp_proj_proj_fk foreign key(p_id) references ddl_proj( p_id)
, e_id  numeric(3)  not null
, constraint ddl_emp_proj_emp_fk foreign key (e_id)references ddl_emp(e_id)
, constraint  ddl_empproj_pk   primary key ( p_id, e_id)
) engine=InnoDB;
```

After running these queries, you will have four tables. The names of the relationship objects are stored in system tables. (user_objects, user_constraints)

Foreign Key

The foreign key constraint specifies the foreign key attribute(s) in the child table and the parent table and column(s) for the relationship. The referenced column(s) in the parent table must have a Unique or Primary Key constraint.

The foreign key constraint is set in the child table. The child claims the parent.

To set a foreign key use a table constraint. Since this is a table constraint you have to indicate the attribute that is the foreign key. You do not have to supply a constraint name

```
CONSTRAINT cnstrName FOREIGN KEY (myCol) REFERENCES parentTbl(col_name)
FOREIGN KEY (myCol) REFERENCES parentTbl(col_name)
```

If the foreign key is made up of multiple columns create a single constraint listing both columns

```
CONSTRAINT cnstrName FOREIGN KEY (myCol1, myCol2)
           REFERENCES parentTbl(col_name1, col_name2 )
```

foreign_key_checks Setting

MySQL has a setting that can be used to turn off foreign key checking. This is not something you would normally want to use.

```
set foreign_key_checks = 0;
set foreign_key_checks = 1;
```

If you set this to 1 which is the default, the foreign key constraints are checked. If you set this to 0 then the foreign key constraints are not checked. This is useful with certain types of inserts that are done as a load.

If you turn foreign keys check off and insert rows those rows are not checked for validity when you turn foreign key checking back on.

You can include a rule as to how to handle deletes with On Delete clause

ON DELETE CASCADE, ON DELETE SET NULL: with the foreign key constraint, you can specify what should happen to the child rows if a parent row is deleted. The default behavior is that you cannot delete a parent row if it has any child rows.

Demo 18: Creates a parent and a child table with cascade delete. These will be very simple tables and I am not naming the constraints.

```
CREATE TABLE  ddl_parent (p_id numeric(3) primary key) engine=InnoDB;
CREATE TABLE  ddl_child  (c_id numeric(3) primary key
                        , fk_id numeric(3)
                        , foreign key(fk_id) references ddl_parent(p_id)
                          on delete cascade) engine=InnoDB;

Insert into ddl_parent(p_id) values (1), (2), (3);
Insert into ddl_child(c_id, fk_id) values (100,2), (101,2), (103,2), (104,3);

Select p_id, c_id, fk_id
from ddl_parent
join ddl_child on ddl_parent.p_id = ddl_child.fk_id;
+------+------+-------+
| p_id | c_id | fk_id |
+------+------+-------+
|    2 |  100 |     2 |
|    2 |  101 |     2 |
|    2 |  103 |     2 |
|    3 |  104 |     3 |
+------+------+-------+
```

Demo 19: deleting rows

```
Delete from ddl_parent where p_id = 2;
```

```
Query OK, 1 row affected (0.02 sec)
```

If I select the rows in the parent table- one row was removed

```
Select * from ddl_parent;
+------+
| p_id |
+------+
|    1 |
|    3 |
+------+
```

If I select the rows in the child table- three rows were removed; even though the feedback from MySQL had indicated only one deleted row.

```
Select * from ddl_child;
+------+-------+
| c_id | fk_id |
+------+-------+
|  104 |     3 |
+------+-------+
```

Demo 20: You can demonstrate this with the set null constraint. Clean up the tables and reinsert the rows.

```
Drop TABLE  ddl_child ;
CREATE TABLE  ddl_child  (c_id numeric(3) primary key
                        , fk_id numeric(3)
                        , foreign key(fk_id) references ddl_parent(p_id)
                          on delete set null) engine=InnoDB;

Insert into ddl_parent(p_id) values (2);
Insert into ddl_child(c_id, fk_id) values (100,2);
Insert into ddl_child(c_id, fk_id) values (101,2);
Insert into ddl_child(c_id, fk_id) values (103,2);
Insert into ddl_child(c_id, fk_id) values (104,3);
```

```
   Delete from ddl_parent where p_id = 2;
```
```
Query OK, 1 row affected (0.02 sec)
```

Now when you look at the rows in the child table the rows are still there but the fk_id field for value 2 are nulled out. This also requires that the fk_id in the child table was a nullable field.

```
   Select * from ddl_child;
   +------+-------+
   | c_id | fk_id |
   +------+-------+
   |  100 |  NULL |
   |  101 |  NULL |
   |  103 |  NULL |
   |  104 |     3 |
   +------+-------+

   Drop TABLE  ddl_child ;
   Drop TABLE  ddl_parent ;
```

You can also use on update set null and on update cascade.

5.3. Other constraint types

DEFAULT: This declaration is used to specify a default value that will be placed in a column.

MyCol DataType Default default_value

Demo 21:

```
   Create table ddl_default (
       id numeric(3)
     , d_state char(2) default 'CA'
   ) engine=InnoDB;
```

You can use the word default in the insert or skip the column to get the default in the column list.

```
   Insert into ddl_default (id, d_state) values (1, 'PA');
   Insert into ddl_default (id, d_state) values (2, 'CA');
   Insert into ddl_default (id, d_state) values (3, default);
   Insert into ddl_default (id)          values (4);
   Select * from ddl_default;
   +------+---------+
   | id   | d_state |
   +------+---------+
   |    1 | PA      |
   |    2 | CA      |
   |    3 | CA      |
   |    4 | CA      |
   +------+---------+
   Drop table ddl_default;
```

CHECK: use this to set other types of tests that a value must meet to be allowed into the table. Before we discuss these you need to know that as of MySQL 5, MySQL does not enforce these check constraints. It lets you include them in your create table statements to allow compatibility with other dbms but they are not enforced. We will look at some examples so that you will recognize this syntax when you see it.

The CHECK constraint condition must evaluate to TRUE or NULL to be satisfied. If you create this as a table constraint, then it can refer to other values in the same column-
such as fld1 <= fld2. If you declare this as a column constraint, then it can refer only to this column.

The syntax for a check constraint is

CONSTRAINT *MyConstraintName* CHECK (*test*)

Use the full test expression even if you are declaring this as a column constraint. (List_Price > 15) The test must be enclosed in parentheses.

Demo 22: Creating a table with a check constraint

```
Create table ddl_check (
    id numeric(3)
  , d_state char(2)
  , constraint dstated_ck
                    check(d_state in ('CA','NV','IL'))
) engine=InnoDB;
```

The first two inserts work.

```
Insert into ddl_check (id, d_state) values (1, 'CA');
Insert into ddl_check (id, d_state) values (2, 'IL');
```

The following insert should fail because NY is not in the approved list of values but **MySQL does not enforce check constraints.** This is something you need to remember when you design a system. MySQL assumes that you will check these rules at the application level.

```
Insert into ddl_check (id, d_state) values (3, 'NY');
```

The following inserts succeeds. The rule for tests in inserts is different than the rule in Where clauses. If the value to be inserted does not specifically fail the test then it is allowed.

```
Insert into ddl_check (id, d_state) values (4, null);

select * from ddl_check;   -- Note that NY was accepted in to the table.
+------+---------+
| id   | d_state |
+------+---------+
|    1 | CA      |
|    2 | IL      |
|    3 | NY      |
|    4 | NULL    |
+------+---------+

show create table ddl_check\G
```

Note that in the create table that MySQL saved the check constraint was not included.

```
*************************** 1. row ***************************
       Table: ddl_check
Create Table: CREATE TABLE `ddl_check` (
  `id` decimal(3,0) DEFAULT NULL,
  `d_state` char(2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

Drop table ddl_check;
```

Demo 23: This failure to implement check constraints applies to other types of check constraints. But is it a good idea to see how a check constraint should be written even if mysql does not (yet?) implement them.

```
Create table ddl_check (
    id numeric(3)
  , weight numeric (3)
  , constraint weight_ck
                    check(weight between 0 and 500)
) engine=InnoDB;
Insert into ddl_check values (1, 450);
Insert into ddl_check values (2, 950);

Drop table ddl_check;
```

Demo 24: Other constraint examples.

```
CREATE TABLE  ddl_dept_2
 (
    d_id    numeric(3)  primary key
  , d_name  varchar(15) not null
  , d_city  varchar(15) not null
  , d_state char(2)
  , d_phone varchar(10)
  , constraint dphone2_un  unique(d_phone)
  , constraint  dlocation2_un unique( d_name, d_city, d_state)
 ) engine=InnoDB;
```

Demo 25: Other constraint examples.

```
CREATE TABLE ddl_emp_2
 (
    e_id     numeric(3)
  , e_name   varchar(15)      not null
  , d_id     numeric(3)
  , salary   numeric(5)
             default 30000
  , hiredate date
  , startsalary numeric(5)
  , constraint emp2_pk  primary key (e_id )
  , constraint emp2_dept2_fk foreign key (d_id) references ddl_dept_2(d_id)
) engine=InnoDB;
```

5.4. Drop restriction

Having relationships created between tables, limits your ability to drop tables. You cannot drop a parent table if there is a related child table.

```
Drop table ddl_dept;
```

```
ERROR 1217 (23000): Cannot delete or update a parent row: a foreign key constraint fails
```

6. Data types you can use in tables

For our tables we have been using only a few types of data to define our columns. MyServer supports more data types and you can find information about all of these in the Van Lans book pages 496-510. The following is a description of some of the most commonly used types. I am not going to give you a lot of ranges for the types- you can look those up if you need them.

6.1. Character strings

CHAR is used for fixed length strings. We use CHAR (2) to store state abbreviations since they all have a 2 letter. If you are storing data where all of the data has the same number of characters- such as SSN, ISBN13, some product codes, then CHAR is appropriate. If necessary the data will be end-padded with blanks to the stated length when stored.

However with MySQL, any trailing blanks in a char attribute are removed when the values are retrieved. Suppose you have an attribute defined as a char (5). You insert into this attribute the value 'cat'. The value is stored as 'cat ' but if you display this attribute in a Select statement, the value display is 'cat' ; if you display the length of the attribute for the row storing 'cat ', it displays a length of 3. This is not a common behavior across different dbms, so you need to watch out for this.

VARCHAR is used for variable length strings. We could use VARCHAR (25) to store customer last names assuming we won't need to store a name longer than 25 characters.

For both of these - if you are defining a table column with char or varchar and try to insert a value longer than the defined length, you will get an error. Char defaults to a length of 1 if you do not state a length; varchar requires a length.

6.2. Integers

We usually use INT which will store number between approximately +-2,000,000, 000.

There are other integer types which have different ranges: form the smallest to the largest

tinyint (-128 to +127) ;smallint; mediumint, int; bigint

When you define a column with an integer type you can include a width- such as Int(5). My SQL refers to this as a display width; it does not limit the range of the values that can be stored. You can store the number 1234567 in an int(5) attribute. The display width might be used by a client or application program for deciding on how wide a column to use for the display.

MySQL also has something called a data type option- an additional bit of information you can add to a data type when declaring a table. Suppose you want a tinyint column but you do not want any negative values to be stored. You can declare the column as tinyint unsigned. This means that negative values cannot be inserted and also the range has shifted. A tinyint range is -128 to +127 the range for a tinyint unsigned is 0 to 255.

You can specify unsigned with any of the numeric types; the range shift occurs with the integer types.

6.3. Fixed precision

Decimal, numeric this lets you set up a type such as decimal(6,2) which can hold numbers from -9999.99 to +9999.99. The first value is the number of digits and the second the number of digits after the decimal point

6.4. Floating point/ Approximate numbers

Float / Double precision- Use this type for numbers where the number of digits after the decimal is not fixed. Suppose you wanted a table of different types of animals and their approximate weight. An elephant weights about 6800 kg and an ant about 0.000003 kg. It would make more sense to use a float than a decimal. We would not say that we are storing these values with up to 6 digits of accuracy after the decimal; we might just be storing these with 2 digits of accuracy.

```
create table weights (an_type varchar(15), an_weight_kg  float);
insert into weights values ( 'ant', 0.000003);
insert into weights values ( 'elephant', 6800);
```

```
select * from weights;
+----------+--------------+
| an_type  | an_weight_kg |
+----------+--------------+
| ant      |     0.000003 |
| elephant |         6800 |
+----------+--------------+
```

6.5. Temporal data

These are for storing date and time values; the range for the dates is '1000-01-01' to '9999-12-31'

DATE	no time component
DATETIME	the time component is hours: minutes: seconds such as `23:59:59`
YEAR	with a range of 1901 to 2155; there is a year(2) version for 2 digit years- hopefully you are not using 2 digit year values.
TIME	time only

MySQL allows various types of none standard date entries. This can get quite complex since it is affected by setting (sql_)mode) that are in effect. We do not discuss sql_mode in this class- but these are some things to watch out for. Depending on the sql_mode, you may be able to enter date such as

```
0000-00-00
2025-00-00
2015-02-31
```

These types of dates can be useful in some applications but may cause problems in other. If you are storing these types of dates pay attention to what is returned by the various date functions.