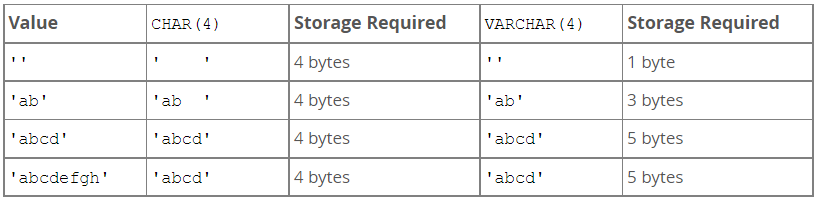
In this section we’ll be revisiting and exploring more datatypes that we can use in SQL, beyond INT and VARCHAR. We’ll put particular focus on dates, times, and timestamps

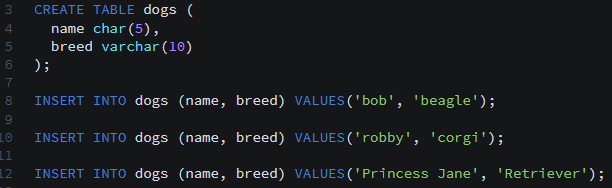
Section slides: <http://webdev.slides.com/coltsteele/mysql-99-102>

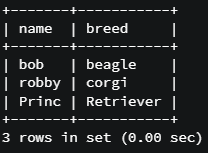
CHAR and VARCHAR

* There are many different datatypes in MySQL, with some being quite similar to each other
* They are broken down into broad categories
  + Storing text
  + Storing numbers
  + Storing dates and times
* The focus on this section is storing text
* Similar to VARCHAR, the **CHAR** datatype is also used to store text. However, it has a **fixed length**
  + Any instance of that datatype must be that declared length
  + Longer text will be truncated
  + Shorter text will be right-padded with spaces
  + Length of CHAR can be declared to be any value from 0 to 255
  + Padded trailing spaces are removed when retrieving data until PAD\_CHAR\_TO\_FULL\_LENGTH SQL mode is enabled
  + It does not have the variable flexibility of VARCHAR
* CHAR is computationally faster than VARCHAR for fixed-length text
  + For example, state abbreviations (2), Y/N flags, or gender (M/F)
* VARCHAR/CHAR comparison table
  + There are minute differences on how CHAR and VARCHAR are stored, but they are typically inconsequential to the functioning of your database unless you have a very large dataset



* CHAR and VARCHAR in practice – we’ll create a table and illustrate the consequences of CHAR and VARCHAR





* + Notice that because “Princess Jane” is longer than 5 characters, the name is truncated
  + When selecting data, we cannot see the tailing spaces (e.g. in “bob”)
  + Remember that VARCHAR will also truncate after the number of characters exceeds the declared number of characters
* Code summary

CREATE TABLE dogs (name CHAR(5), breed VARCHAR(10));

INSERT INTO dogs (name, breed) VALUES ('bob', 'beagle');

INSERT INTO dogs (name, breed) VALUES ('robby', 'corgi');

INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retriever');

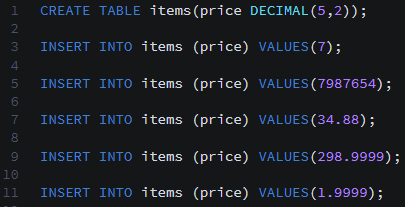
SELECT \* FROM dogs;

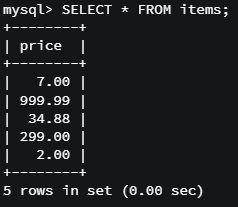
INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retrievesadfdsafdasfsafr');

SELECT \* FROM dogs;

DECIMAL Datatype

* There are MANY number datatypes in MySQL. We’ve already seen INT, which works with whole numbers
* DECIMAL is a datatype that can include a decimal point and decimal numbers
  + Documentation: <https://dev.mysql.com/doc/refman/5.7/en/precision-math-decimal-characteristics.html>
* DECIMAL is declared with two numbers as follows: DECIMAL(M, D)
  + *M* is the total number of digits that our number can have (includes numbers before AND after the decimal point)
  + *D* is the number of digits allowed after the decimal place. Clearly, D ≤ M
  + There may be no more than 30 digits after the decimal point, so D ≤ 30
* Example of DECIMAL





* + A few items to note here:
    - Two decimals were added to the end of the whole number 7, to make it 7.00
    - When we entered 7987654, we exceeded the maximum number of digits allowed by the decimal, which is 5. So the system returned 999.99, the largest number it could within constraints
    - 298.9999 was rounded to two decimal places, to 299.00
    - 1.9999 was rounded to two decimal places, to 2.00
* Code summary

CREATE TABLE items(price DECIMAL(5,2));

INSERT INTO items(price) VALUES(7);

INSERT INTO items(price) VALUES(7987654);

INSERT INTO items(price) VALUES(34.88);

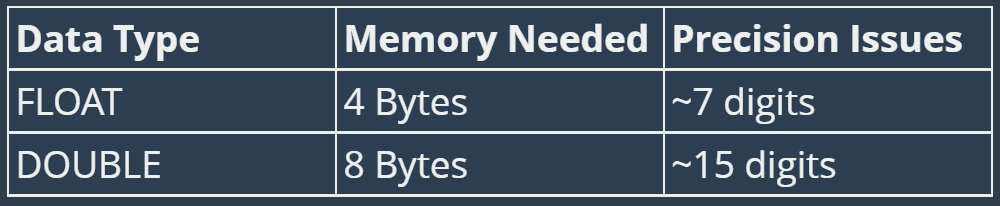
INSERT INTO items(price) VALUES(298.9999);

INSERT INTO items(price) VALUES(1.9999);

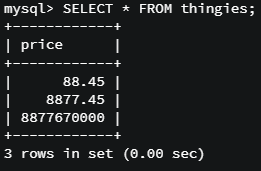
SELECT \* FROM items;

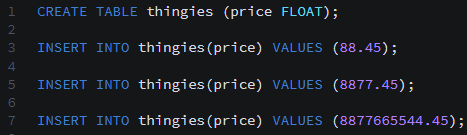
FLOAT and DOUBLE

* FLOAT and DOUBLE are datatypes that are also used to handle decimal numbers
  + The differences between DECIMAL, DOUBLE, and FLOAT are quite technical and involve how values are stored in memory
  + Practically speaking, these differences are not of concern, so long as you understand the *functional* differences between them
  + Practical differences (see documentation here: <https://dev.mysql.com/doc/refman/5.7/en/precision-math-numbers.html>):
    - Decimals are fixed-point and calculations are exact
    - FLOATs and DOUBLEs are floating point and calculations are approximate
* Why should we use FLOAT and DOUBLE if they are approximate?
  + They store larger numbers using less space in memory, but at the cost of precision
  + This does not mean that FLOAT and DOUBLE are wildly inaccurate, but they will not be as precise as DECIMAL
  + FLOAT and DOUBLE both start to have precision issues at around a certain number of digits. See the table below:



* So, which numeric datatypes should you use when handling information that involves decimals? The instructor recommends that we **use DECIMAL when handling decimal numbers *unless* you are confident that precision is not critical**
* Illustration of the imprecision of FLOAT/DOUBLE





* + Note how the last value entered resolved to 8877670000, which is considerably different from 8877665544.45. This illustrates just how precision can be lost when using FLOAT or DOUBLE
* Code summary

CREATE TABLE thingies (price FLOAT);

INSERT INTO thingies(price) VALUES (88.45);

SELECT \* FROM thingies;

INSERT INTO thingies(price) VALUES (8877.45);

SELECT \* FROM thingies;

INSERT INTO thingies(price) VALUES (8877665544.45);

SELECT \* FROM thingies;