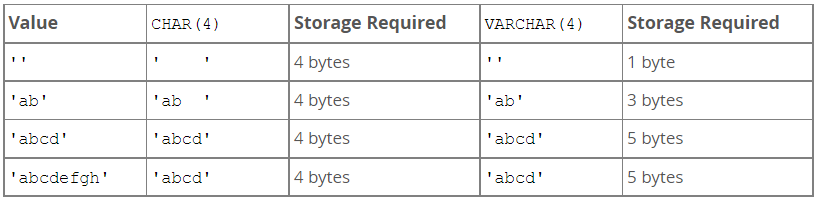
In this section we’ll be revisiting and exploring more datatypes that we can use in SQL, beyond INT and VARCHAR. We’ll put particular focus on dates, times, and timestamps

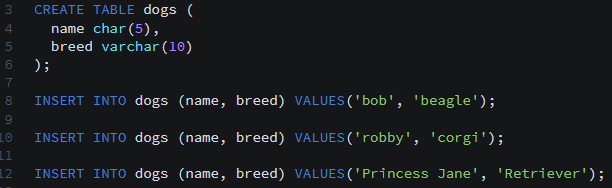
Section slides: <http://webdev.slides.com/coltsteele/mysql-99-102>

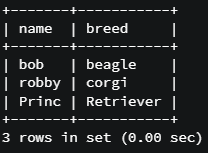
CHAR and VARCHAR

* 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');

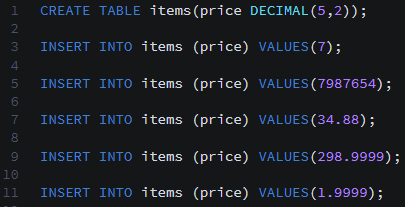
SELECT \* FROM dogs;

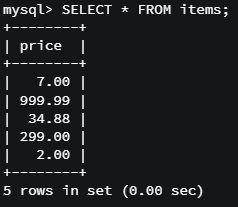
INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retrievesadfdsafdasfsafr');

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);

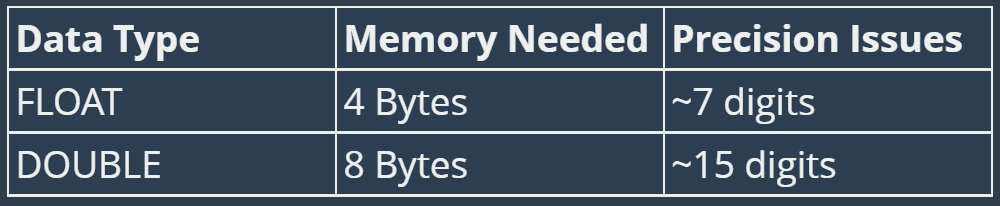
INSERT INTO items(price) VALUES(298.9999);

INSERT INTO items(price) VALUES(1.9999);

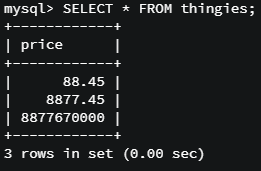
SELECT \* FROM items;

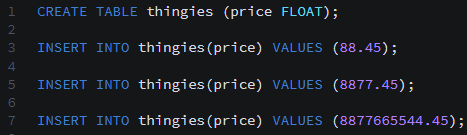
FLOAT and DOUBLE

* 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);

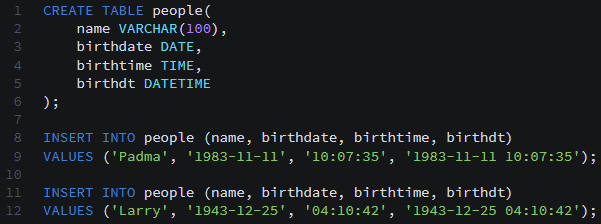
SELECT \* FROM thingies;

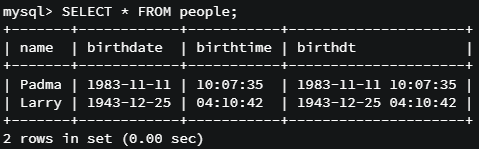
DATE, TIME, and DATETIME

* The DATE datatype is designed for storing a date *without* the time
  + Format: ‘YYYY-MM-DD’ (includes quotes)
* The TIME datatype stores the time, but no date
  + Not used that frequently on its own, because storing time without a date is not very common
  + Format: “HH:MM:SS”
* The DATETIME datatype stores values with a date and a time. You get both!
  + Used *very* frequently
  + A common use case is for storing the date and time of when a new row/entry is entered into a database table
  + Format: ‘YYYY-MM-DD HH:MM:SS’

Creating DATE Data

* Let’s create a new table and use all three of these datatypes
  + We’ll be using this data in the next few videos to go over several useful datetime functions!





* Code summary

CREATE TABLE people (name VARCHAR(100), birthdate DATE, birthtime TIME, birthdt DATETIME);

INSERT INTO people (name, birthdate, birthtime, birthdt)

VALUES('Padma', '1983-11-11', '10:07:35', '1983-11-11 10:07:35');

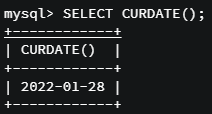
INSERT INTO people (name, birthdate, birthtime, birthdt)

VALUES('Larry', '1943-12-25', '04:10:42', '1943-12-25 04:10:42');

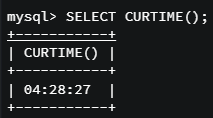
SELECT \* FROM people;

CURDATE, CURTIME, and NOW

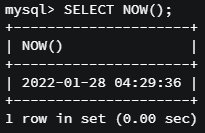
* **CURDATE** give us the current date



* **CURTIME** gives us the current time

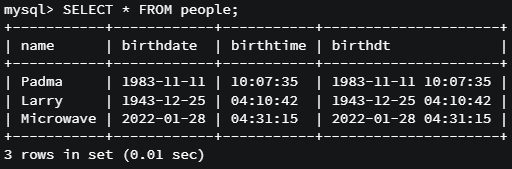


* **NOW** gives us the current date and time as a DATETIME



* These functions are not particularly useful when used in isolation, but become very useful when used with INSERT statements. Let’s add someone to our *people* table with a birthdate of right now



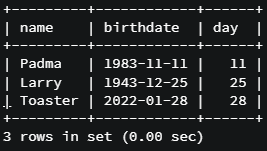


* We’ll also use these functions for date arithmetic, as we’ll see later

Formatting Dates

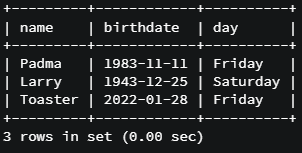
* Dates, times, and datetimes are stored in a format that’s not particularly pleasing to look at. Thankfully, there’s a way to convert this into a much more human-readable format
* Documentation: <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>
  + The instructor uses very few of these
* We can the following functions to get some information from our datetimes
  + **DAY()** returns the day of the month (integer)





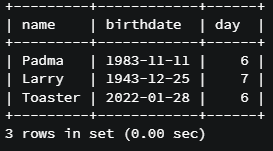
* + **DAYNAME()** returns the name of the day of the week





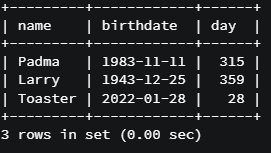
* + **DAYOFWEEK()** returns the number of the day of the week (Sunday = 1, Saturday = 7)





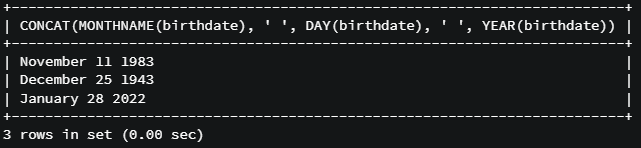
* + **DAYOFYEAR()** returns the number of the day of the year (accounts for leap years)





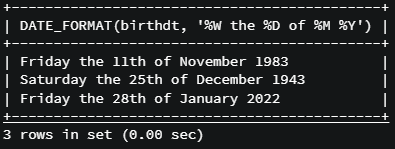
* Other functions include:
  + MONTH() – gives the number of the month
  + MONTHNAME() gives the name of the month
  + HOUR() gives the hour of the time or datetime
* With these functions, we can do things like say “April 212017”.
  + One way to do this is with a long CONCAT() statement





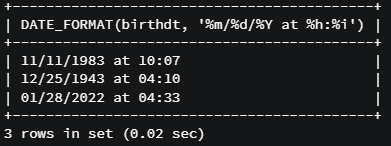
* + But there is another (better) way, which is the use the **DATE\_FORMAT()** function (<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_date-format>), which allow us to specify what we want from a date
    - Includes specifiers that allow you to designate what you want out of the datetime object
    - You pass in the date and the specifier string and get your formatted date back!
    - Very useful function for formatting dates and times that does essentially everything that those individual functions do!
    - One example for human-readable date





* + - Another example using time





* Code summary

SELECT name, birthdate FROM people;

SELECT name, DAY(birthdate) FROM people;

SELECT name, birthdate, DAY(birthdate) FROM people;

SELECT name, birthdate, DAYNAME(birthdate) FROM people;

SELECT name, birthdate, DAYOFWEEK(birthdate) FROM people;

SELECT name, birthdate, DAYOFYEAR(birthdate) FROM people;

SELECT name, birthtime, DAYOFYEAR(birthtime) FROM people;

SELECT name, birthdt, DAYOFYEAR(birthdt) FROM people;

SELECT name, birthdt, MONTH(birthdt) FROM people;

SELECT name, birthdt, MONTHNAME(birthdt) FROM people;

SELECT name, birthtime, HOUR(birthtime) FROM people;

SELECT name, birthtime, MINUTE(birthtime) FROM people;

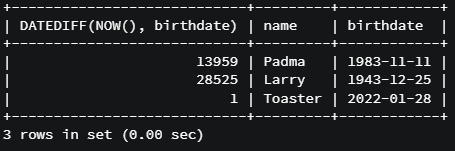
SELECT CONCAT(MONTHNAME(birthdate), ' ', DAY(birthdate), ' ', YEAR(birthdate)) FROM people;

SELECT DATE\_FORMAT(birthdt, 'Was born on a %W') FROM people;

Date Mathematics

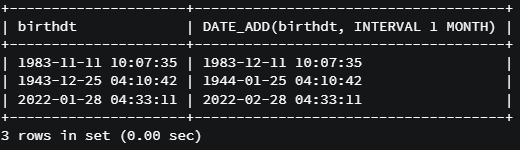
* We are able to do things like add days to dates, find out how many days are between two dates, add time to a datetime, etc.
  + This is common in things like forum comments and blog posts where you want to know how long ago something happened
* **DATEDIFF** calculates the difference between two dates that are passed into the function
  + <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_datediff>
* Let’s go back to our *people* table and calculation how many days *ago* each person was born





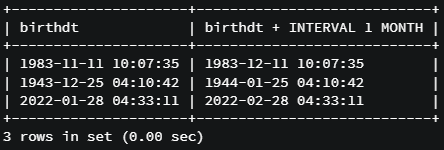
* **DATE\_ADD()** takes a provided date and a second argument where we specify using an interval keyword describing what we want to add
  + <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_date-add>
* Let’s go to our *people* table and add 1 month to all of their birth datetimes





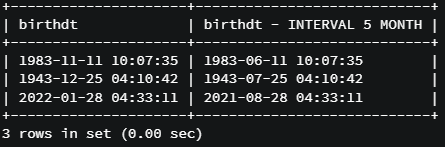
* Date arithmetic can also be performed with “+” and “-“ signs without using the DATE\_ADD() function
  + Syntax: *date* +/- INTERVAL exp. Unit
  + Example: Adding a month





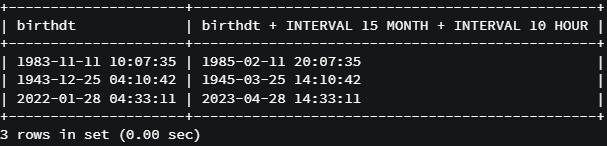
* + Example: subtracting months





* + Chaining together date math operations





* Practically speaking, DATE\_DIFF() tends to be used more often as it is oftentimes important to calculate how much time has passed since a given timestamp
* Code summary

SELECT \* FROM people;

SELECT DATEDIFF(NOW(), birthdate) FROM people;

SELECT name, birthdate, DATEDIFF(NOW(), birthdate) FROM people;

SELECT birthdt FROM people;

SELECT birthdt, DATE\_ADD(birthdt, INTERVAL 1 MONTH) FROM people;

SELECT birthdt, DATE\_ADD(birthdt, INTERVAL 10 SECOND) FROM people;

SELECT birthdt, DATE\_ADD(birthdt, INTERVAL 3 QUARTER) FROM people;

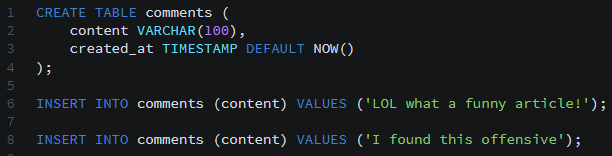
SELECT birthdt, birthdt + INTERVAL 1 MONTH FROM people;

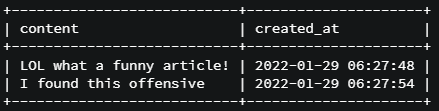
SELECT birthdt, birthdt - INTERVAL 5 MONTH FROM people;

SELECT birthdt, birthdt + INTERVAL 15 MONTH + INTERVAL 10 HOUR FROM people;

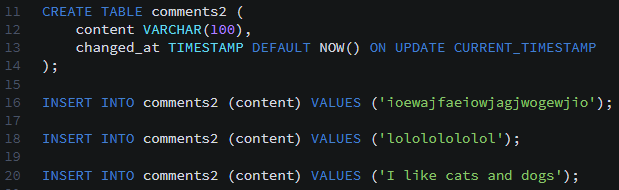
Working with TIMESTAMPS

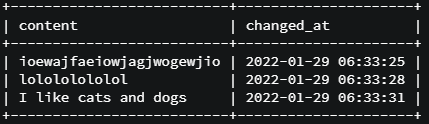
* Timestamps are used to store information metadata regarding *when* something is created or modified. In MySQL, TIMESTAMP is also a datatype!
  + Documentation: <https://dev.mysql.com/doc/refman/8.0/en/datetime.html>
* TIMESTAMP and DATETIME both contain information regarding date and time. However, they are different in the range of dates and times that they support.
  + TIMESTAMP only goes from 1970 to 2038
  + DATETIME supports ranges from the year 1000 to the year 9999
* Practically speaking, we don’t need to worry about these ranges when we’re making timestamps
  + Why does TIMESTAMP exist when we already have DATETIME? It’s mainly because TIMESTAMP consumes less space in memory compared to DATETIME
* Main takeaway: Use DATETIME for everything EXCEPT for when you actually want to create timestamps
* Example: In the code below, we create a table called *comments* with a column “created\_at” whose datatype is TIMESTAMP
  + We set the default value of “created\_at” as NOW(). This will cause any data we insert to be accompanied by a timestamp of the exact time it was inserted
  + We’ll also insert some comments. We do not need to insert values for “created\_at” because the value will be defaulted



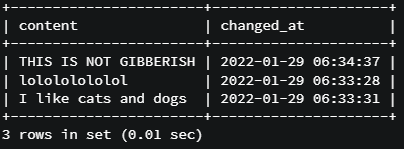


* + You can also order your data by TIMESTAMPS
* What if we want to store a TIMESTAMP of when data is edited or changed? We can do that as well. We need to modify the datatype declaration in the table to reflect that we want a CURRENT\_TIMESTAMP upon an update to the data
  + We can also use NOW() instead of CURRENT\_TIMESTAMP









* Code summary

CREATE TABLE comments (

content VARCHAR(100),

created\_at TIMESTAMP DEFAULT NOW()

);

INSERT INTO comments (content) VALUES('lol what a funny article');

INSERT INTO comments (content) VALUES('I found this offensive');

INSERT INTO comments (content) VALUES('Ifasfsadfsadfsad');

SELECT \* FROM comments ORDER BY created\_at DESC;

CREATE TABLE comments2 (

content VARCHAR(100),

changed\_at TIMESTAMP DEFAULT NOW() ON UPDATE CURRENT\_TIMESTAMP

);

INSERT INTO comments2 (content) VALUES('dasdasdasd');

INSERT INTO comments2 (content) VALUES('lololololo');

INSERT INTO comments2 (content) VALUES('I LIKE CATS AND DOGS');

UPDATE comments2 SET content='THIS IS NOT GIBBERISH' WHERE content='dasdasdasd';

SELECT \* FROM comments2;

SELECT \* FROM comments2 ORDER BY changed\_at;

CREATE TABLE comments2 (

content VARCHAR(100),

changed\_at TIMESTAMP DEFAULT NOW() ON UPDATE NOW()

);