Data types in MySQL - Text lecture
https://en.wikipedia.org/wiki/SQL#Data_types

Each column has a data type. They define what type of data is allowed in each field. Understanding them unlocks tons of functions and features and avoids pitfalls that cause a lot of heartache. These can be slightly different between SQL environments, but they all accomplish the same things.

VARCHAR(n) where N is the maximum length of the string (65,535 characters)
CHAR(n) fixed length of the string regardless of contents. It will be padded with spaces. Anything longer than a few characters should be a VARCHAR as it generally takes up less space in the database.
Read more: http://dev.mysql.com/doc/refman/5.7/en/char.html

Each data type has a different size and takes different amounts of time to process. Strings are handy in that they don't care what they are holding, dates, numbers, true/false, but a database that relies on strings when they could use other data types are bigger, slower, inefficient and miss out on all of the features built into MySQL.

Date - no consideration of time. Usually not the one you pick.
Time - also not the one you usually want because of how limited it is. Think of how you might handle midnight
Timestamp - likely always the one you want to define.
Int - Integer is plus or minus -2,147,483,648 so it can handle basically anything. There are other types, but int is usually good enough in most cases.

Boolean - unlike other database environments, MySQL doesn't currently have a boolean (True/False). Instead we use tinyint(1). When creating a table, you can use "boolean", and MySQL will use tinyint(1) instead. 0 = "False", and 1="True".

Decimal - Exact Values like money and measurements. Decimals are defined as follows DECIMAL(A,B) where A is the total number of digits, and B is the number of digtis after the period. For example DECIMAL(5,2) could handle anything from -999.99 to 999.99. It could not handle 1000, or 123.456.

Where and not equal - Text lecture

In section 2 we looked at where but didn't dive very far into it. Now we will explore more options of the Where Clause….

Where is how we control which rows are returned.

Example:

```
SELECT Title
FROM movies
WHERE 1=1;
```

Always true, so return every title for all movies(rows) in the table.

```
SELECT 1 As Result
FROM movies
WHERE 1=1;
SELECT 1 As Result
FROM movies
WHERE Rating='PG';
```

It's important to see that WHERE only controls which rows are returned, but has no other effect on what data actually returns.

```
SELECT *
FROM movies
WHERE RATING = "PG"
 AND Title LIKE "Night%";
```

You can string as may conditions together as you need using AND

```
SELECT *
FROM movies
WHERE RATING = "PG"
 AND Title NOT LIKE "Night%";
```

NOT allows you to define what you don't want displayed.

Comparison operators - Text lecture

Greater than and less than are pretty straight forward in their use and meaning:

```
SELECT title
    ,release_year
FROM movies
WHERE release_year > 2005;
```
Notice that Night at the Museum and National Treasure are not in the list.

```
SELECT title
    ,release_year
FROM movies
WHERE release_year > 2005
 AND  release_year < 2016;
```
We can combine greater than and less than to limit to a specific range.

We can include the limits of the range using >= or <= (Greater Than or Equal To/Less Than or Equal To)
```
SELECT title
    ,release_year
FROM movies
WHERE release_year >= 2005
 AND  release_year <= 2016;
```
While this is a great way to accomplish this, it's also perfectly acceptable to use BETWEEN

```
SELECT title
    ,release_year
FROM movies
WHERE release_year BETWEEN 2005 AND 2016;
```
Exact Same results that include 2005 and 2016

Combination of both Greater Than and Less Than is <> which means Not Equal.
```
SELECT title
      ,release_year
FROM movies
WHERE release_year <> 2016;
```
Notice that the only value missing is the 2016

Not Equal also works well with other data types like strings.
```
SELECT title
```

```
    ,release_year
FROM movies
WHERE title <> 'Zoolander';
```
Just like = and <>, we can use IN to include or exclude multiple values at the same time
```
SELECT title
    ,release_year
FROM movies
WHERE release_year IN (2005,2009,2010);
```
In order to exclude items, we can use NOT IN
```
SELECT title
    ,release_year
FROM movies
WHERE release_year NOT IN (2005,2009,2010);
```
This also works well for Strings
```
SELECT title
    ,release_year
FROM movies
WHERE title IN ('Zoolander','Tropic Thunder');
```

OR Will allow you to have distinct conditions that may otherwise conflict

example:
```
SELECT *
FROM movies
WHERE RATING = "R"
  OR Title LIKE "Zoo%";
```
Both of these accomplish the same thing but ()s and spacing help avoid mistakes.

```
SELECT *
FROM movies
WHERE ( RATING = "R"
       OR Title LIKE "Zoo%");
```

Null and blanks - Text lecture

Let's add more data to help explore the issue of null and empty string:

```
INSERT INTO movies (title, rating)
VALUES
  ('DodgeBall: A True Underdog Story' ,'PG-13')
  ,('Along Came Polly','PG-13')
  ,('Anchorman:The Legend of Ron Burgundy','PG-13');
INSERT INTO movies (title, release_year)
VALUES
  ('Anchorman 2: The Legend Continues' ,2013)
  ,('Megamind',2010);
INSERT INTO movies (title, rating)
VALUES('Fight Club','');
```

NULL Means there is no data. This happens all the time depending on the data you are dealing with. The issue comes is when you are trying come compare a non-existent value (null) to anything

```
SELECT *
FROM movies
WHERE NOT rating = 'PG-13'
 AND  NOT rating = 'PG'
 AND  NOT rating = 'R';
```

Where did Anchorman 2 and Megamind go?

```
SELECT *
FROM movies
WHERE rating IS NULL;
```

The issue is we were comparing NULL (a non existent value) to a value. Non-existent values don't equal anything. “IS NULL” and IS NOT NULL” compares the field in a way that will return content in these scenarios.

```
SELECT *
FROM movies
WHERE rating = 'R'
      OR rating IS NULL;
```

We can compare a string to an empty string.

```
SELECT *
FROM movies
```

```
WHERE rating = 'R'
    OR rating = ‘’;
SELECT *
FROM movies
WHERE rating = 'R'
    OR IFNULL(rating,'') = '';
```

IFNULL(A, B) where A is the field to be compared and B is the default value to be used when NULL is found. Oracle uses NVL and SQL Server uses ISNULL

Case statements - Text lecture

```
SELECT title
    ,rating
FROM movies;
SELECT title
    ,rating
    ,CASE
      WHEN RATING = 'PG'    THEN 'Bring the Kids!'
      WHEN RATING = 'PG-13' THEN 'Older Kids'
      WHEN RATING = 'R'     THEN 'Not for Kids'
      ELSE 'No Information'
     END AS AUDIENCE
FROM movies;
```

Case statements allow a lot of flexibility. They evaluate the first WHEN, and if it passes, it uses the THEN for the return value and skips to the next row. Else option and allows for a catch-all value like “No Information”
The different 'whens' can be inspecting different values, and the CASE Statment can be used in the SELECT or the WHERE.
Let revisit the NULL query from earlier.

```
SELECT *
FROM movies
WHERE CASE
      WHEN rating IS NULL THEN 1
      WHEN rating = ''    THEN 1
      WHEN release_year <= 2007 THEN 1
    END = 1;
```

Dates and times - Text lecture

Some time functions:

```
SELECT NOW()
    ,CURDATE()
    ,CURTIME();
```

More information on time functons:
http://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html

MySQL has prebuilt functions that offer some handy functionality when dealing with dates and times.

NOW() returns the system date and time of the server
CURDATE() returns the system date without the time
CURTIME() returns the system time without the date
NOW() is useful for adding a timestamp when a row is being added to the database, but let's use it to explore some other data functions

```
SELECT NOW()
    ,CURDATE()
    ,CURTIME()
    ,YEAR(NOW())
    ,YEAR(CURDATE());
```

We can have just the year pulled out. This function could easily be used to help group rows by years

```
SELECT NOW()
    ,MONTH(NOW())
    ,MONTHNAME(Now());
```

MySQL allows you to pull just the month by number or name

```
SELECT NOW()
    ,DAY(NOW())
    ,DAYNAME(Now())
    ,DAYOFMONTH(NOW())
    ,DAYOFWEEK(NOW())
    ,DAYOFYEAR(NOW());
```

DAY gives the day of the month
DAYNAME gives the name of the day of the week

DAYOFMONTH is the same as DAY
DAYOFWEEK gives the number corresponding to the day. Sunday is 1 and Saturday is 7
DAYOFYEAR gives the day if we were counting from Jan 1st as 1,
You may have noticed that your cloud 9 Environment is not in your current timezone. Instead the System time is UTC.
Find your timezone here:
https://en.wikipedia.org/wiki/List_of_tz_database_time_zones
Then you can set your timezone:
SET time_zone = '-7:00';
SET GLOBAL time_zone = '-7:00';
Then you can check the time difference:
select @@global.time_zone, @@session.time_zone;
You can then check your newly set time:
select NOW();

Date, time and math - Text lecture
Let's do some math with dates
Start with a table

```
CREATE TABLE person (personID INT NOT NULL AUTO_INCREMENT
                    ,FNAME varchar(50) DEFAULT NULL
                    ,LNAME varchar(50) DEFAULT NULL
                    ,dob date DEFAULT NULL
                    ,PRIMARY KEY (personID));
```

Insert a couple of rows

```
INSERT INTO person (FNAME, LNAME,dob)
VALUES ('Mashrur', 'Hossain','1981-12-25')
     ,('Mark', 'Futre','1985-01-03');
```

Let's find ages, so we start with

```
SELECT dob AS Birthdays
     ,CURDATE()   AS Today
FROM person;
```

MySQL needs to know that you want to do date math and not just regular math. There are rules for dates like 12 months in a year, and leap years and days in a week. We need to use date specific functions to apply these rules like TIMESTAMPDIFF()

```
SELECT dob AS "Birthday"
     ,CURDATE()   AS Today
     ,TIMESTAMPDIFF(YEAR,dob,CURDATE()) AS AGE
FROM person;
```

Let's find when when the next birthday will be. In this case we are taking the year only from the current date, attaching it to the month/day of the dob and displaying it as 'Birthday This Year'. Similarly, we are taking the current date, adding a year, and then attaching the dob month/day to calculate what the dob will be next year. The issue is we don't know if it already happened this year or if it's going to take place next year.

First let's start with printing out birthday this year and birthday next year

```
SELECT FNAME
     ,dob AS Birthday
     ,CURDATE()   AS Today
     ,STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())
),'%m/%d/%Y') AS "Birthday This Year"
     ,DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR ) AS "Birthday Next Year"
FROM person;
```

Now, we can use a CASE Statement to help decide which birthday to use:

```
SELECT FNAME
     ,dob AS Birthday
     ,CURDATE()   AS Today
     ,STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())
),'%m/%d/%Y') AS "Birthday This Year"
     ,DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR ) AS "Birthday Next Year"
     ,CASE
        WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y') >= CURDATE()
          THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y')
        ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
```

```
m/%d/%Y'), INTERVAL 1 YEAR )
      END AS "Next Birthday"
FROM person;
```

Text solution to final project on birthday reporting

Section 3 final project!

Create a table with your four best friends first names, last names and their birthdays.
Create a report which will return the following:
1. First and last name together in one column
2. Current Age
3. Create a column - if birthday is today, return ‘Call Today’, else if birthday is less than 14 days, display ‘Send Card”, else display Birthday is in (name of month).
4. Only return friends that are within the next 6 months.
5. Test your report by replacing current date with various hard coded dates.
First create the table and fill in data:

```
CREATE TABLE Friends (FriendID INT NOT NULL AUTO_INCREMENT
                  ,FNAME varchar(50) DEFAULT NULL
                  ,LNAME varchar(50) DEFAULT NULL
                  ,dob date DEFAULT NULL
                  ,PRIMARY KEY (FriendID));
INSERT INTO Friends (FNAME, LNAME,dob)
VALUES ('Mashrur', 'Hossain','1982-12-01')
     ,('Matt', 'Berstein','1980-08-05')
     ,('Anastasia', 'Ivanov','1989-04-01')
     ,('Mark', 'Futre','1989-07-04');
SELECT *
FROM Friends;
```

Its best to build these in phases. Return the individual parts before you incorporate them into case statements

```
SELECT CONCAT(FNAME,' ',LNAME)  AS Friend
     ,TIMESTAMPDIFF(YEAR,dob,CURDATE()) AS AGE
     ,dob
     ,CURDATE()
     ,STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())
```

```
),'%m/%d/%Y') AS BDAY_THIS_YR
      ,DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR ) AS BDAY_NEXT_YR
      ,DATE_ADD(CURDATE(), INTERVAL 6 MONTH) AS 6Months
      ,MONTHNAME(dob) AS "Birth Month"
FROM Friends
WHERE CASE
         WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y') >= CURDATE()
           THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y')
         ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR )
       END BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 6
MONTH)
       ;
Now let's add in an update:
SELECT CONCAT(FNAME,' ',LNAME)  AS Friend
      ,TIMESTAMPDIFF(YEAR,dob,CURDATE()) AS AGE
      ,CASE
         WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y') = CURDATE()
           THEN 'Call Today'
         WHEN CASE
                WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y') >= CURDATE()
                  THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y')
                ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR )
```

```
            END <= DATE_ADD(CURDATE(), INTERVAL 14 DAY)
          THEN 'Send a card'
        ELSE concat('Birthday is in ',MONTHNAME(dob))
      END AS ToDo
FROM Friends
WHERE CASE
        WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y') >= CURDATE()
          THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y')
        ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR(CURDATE())),'%
m/%d/%Y'), INTERVAL 1 YEAR )
      END BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 6
MONTH)
      ;
```

That's it! now let's test it with hardcoding different dates:
Find and replace works wonders.

```
SELECT CONCAT(FNAME,' ',LNAME)  AS Friend
     ,TIMESTAMPDIFF(YEAR,dob,'2016-08-05') AS AGE
     ,CASE
        WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y') = '2016-08-05'
          THEN 'Call Today'
        WHEN CASE
              WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y') >= '2016-08-05'
                THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y')
              ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y'), INTERVAL 1 YEAR )
            END <= DATE_ADD('2016-08-05', INTERVAL 14 DAY)
```

```
          THEN 'Send a card'
        ELSE concat('Birthday is in ',MONTHNAME(dob))
      END AS ToDo
FROM Friends
WHERE CASE
        WHEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y') >= '2016-08-05'
          THEN
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y')
        ELSE DATE_ADD(
STR_TO_DATE(CONCAT(MONTH(dob),'/',DAY(dob),'/',YEAR('2016-08-
05')),'%m/%d/%Y'), INTERVAL 1 YEAR )
      END BETWEEN '2016-08-05' AND DATE_ADD('2016-08-05', INTERVAL 6
MONTH)
      ;
```