Database Server: Service running on a machine (server or desktop), it houses many databases.

Database: Contains all of the persistent information for your applications

Schema: In MySQL, this is synonymous with database and can be used interchangeably, but shouldn’t because SQL Server and Oracle treat them differently.   
SQL Server/Oracle - Schema is a namespace inside the database with different permissions associated with it.

To clear screen from mysql cli -> \! clear

Open MySQL ->  
Type : mysql-ctl cli

Some commands ->

show databases;

show schemas;

SQL statements end with “;”  
If you don’t put the “;”, MySQL will wait to execute until you ad it in.

To create a new database ->  
create database project1;

MySQL is case sensitive.  
Don’t put spaces in the database name.  While possible, it will just cause issues later..  Spaces generally cause annoyances in names of databases, tables, columns and everything.

create schema project1; this statement would accomplish the same thing as the command above to create database

Create another database ->  
create database mashrur1;

show databases;

To delete (or drop) the database mashrur1 (warning: this will immediately delete it) ->  
drop database mashrur1;

To use a specific database for queries we'll be running and impacting (let's say we want this to be project1):  
use project1;

Now when we create and do things like create objects or call objects, database “project1” is assumed.  
Note: up arrow recalls previous commands.

 -Tables are just like Excel Spreadsheets

- Columns have headers and are for certain types of data like strings, dates, numbers etc.  
- Rows are the actual data which all fit inside the column data types.

- Structure matters. Forgetting a comma or a line or a space can break things.

To create a table called people (The capitalization of CREATE TABLE below is to separate out the sql code from the names of the tables, you can also just say create table, capitalization matters in column names):  
CREATE TABLE people ( PersonID int  
                     ,first\_name varchar(100)  
                     ,last\_name  varchar(100)  
);

To show database in use, type in select database();  
show tables;

CREATE TABLE people2 ( PersonID INT NOT NULL AUTO\_INCREMENT  
                      ,FIRST\_NAME VARCHAR(100) NULL  
                      ,LAST\_NAME VARCHAR(100) NULL  
                      ,PRIMARY KEY (PersonID));

Primary Key is a unique value that each row will have, auto\_increment makes it easy by automatically assigning them

If we want to remove a table, use DROP TABLE (or drop table)  
show tables;

To add a column, first see the columns that are there by typing in -> show columns from people2;

Notice there are 3 columns;

To add a column called DOB:  
alter table people2 add column DOB date NULL;

Similarly, to drop a column, type in:  
alter table people2 drop column DOB;     
show columns from people2;

To delete tables:  
drop table people;  
drop table people2;

To add data to tables:

INSERT INTO actors ( FIRST\_NAME , LAST\_NAME )  
VALUES ('Ben','Stiller');

We’ll discuss select further in the next lecture, but to show all data in the table:  
SELECT \*  
FROM project1.actors;

To insert multiple values into actors table:  
INSERT INTO project1.actors ( FIRST\_NAME , LAST\_NAME )  
VALUES ('Owen','Wilson')  
      ,('Christine','Taylor')  
      ,('Will','Ferrell')  
      ,('Milla','Jovovich')  
      ,('Jerry','Stiller')  
      ,('David','Duchovny')  
      ,('Jon','Voight')  
      ,('Nathan','Graham');

Now that we have Zoolander, lets add new actors from Zoolander 2  
INSERT INTO actors ( FIRST\_NAME , LAST\_NAME )  
VALUES ('Kristen','Wiig')  
      ,('Penélope','Cruz')  
      ,('Lenny','Kravitz')  
      ,('Macaulay','Culkin')  
      ,('Justin','Bieber')  
      ,('Cyrus','Arnold');  
     
Now let's add data to the movies table:  
INSERT INTO movies ( Title , ReleaseYear, Rating )  
VALUES ('Zoolander',2001,'PG-13')  
      ,('Zoolander2',2016,'PG-13')  
      ,('Night at the Museum: Secret of the Tomb',2014,'PG')  
      ,('Night at the Museum: Battle of the Smithsonian',2009,'PG')  
      ,('Night at the Museum',2005,'PG')  
      ,('National Treasure',2004,'PG')  
      ,('Tropic Thunder',2008,'R');

**SELECT STATEMENT**

SELECT \* FROM actors;

SELECT \* FROM movies;

-> SELECT section lists the results to be returned.  
\* Means everything  
From is where you list tables.  
We will add lots of extra details but the SELECT \* FROM is the general structure of how we retrieve any and all data from the database.

Listing the columns returns the same results as \*  
SELECT ActorID  
      ,FIRST\_NAME  
      ,LAST\_NAME  
FROM actors;   
    
SELECT FIRST\_NAME  
      ,LAST\_NAME  
FROM actors;     
Returns same number of rows, but only the columns that were requested (FIRST\_NAME AND LAST\_NAME)

**Concat and substring**

CONCAT - Use to add columns together for display

SELECT CONCAT(LAST\_NAME, ', ', FIRST\_NAME)  
FROM actors;

Returns same number of rows, but with our calculated row.

SQL lets you rename columns for display as well:  
SELECT CONCAT(LAST\_NAME, ', ', FIRST\_NAME) AS ACTOR\_NAME  
FROM actors;

To grab certain data from the rows under a column (instead of all the data) you can use substring:  
SELECT LAST\_NAME, SUBSTRING(LAST\_NAME,1,3)  
FROM actors;

A useful query to generate user name from first\_name and last\_name columns:  
SELECT FIRST\_NAME  
      ,LAST\_NAME  
      ,SUBSTRING(FIRST\_NAME,1,1)  
      ,CONCAT(SUBSTRING(FIRST\_NAME,1,1),LAST\_NAME) AS USER\_NAME  
FROM actors;

Select allows you to name columns with spaces using “”, you can create new columns, and calculate columns.  You can reuse columns for basically anything.

SELECT CONCAT(LAST\_NAME, ', ', FIRST\_NAME) AS "Actor's Names"  
      ,LAST\_NAME AS LNAME  
      ,FIRST\_NAME AS "First Name"  
      ,CONCAT(SUBSTRING(FIRST\_NAME,1,1),LAST\_NAME) AS USER\_NAME  
      ,'Any Text' AS Anything  
      ,1+2        AS Math  
FROM actors;

**LIMIT & ORDER BY**

Large datasets where you don’t want to tax the system returning millions of rows use limit:

SELECT \*  
FROM actors  
LIMIT 10;

This query above will return the first 10 results from the actors table instead of all the rows.

Order by can be used to sort the data that is returned in the query result:  
SELECT \*  
FROM actors  
ORDER BY 3;

This query above will return the results sorted by the 3rd column of the table in ascending order by default

Which is the same as   
SELECT \*  
FROM actors  
ORDER BY LAST\_NAME ASC;

The ASC is for ascending which is implied.  The opposite is descending DESC

SELECT \*  
FROM actors  
ORDER BY 3 DESC;

We can also sort by multiple columns at the same time and by name  
SELECT \*  
FROM actors  
ORDER BY LAST\_NAME ASC, FIRST\_NAME DESC;  
Notice the order the Jerry and Ben are in.

And you can combine LIMIT and ORDER BY  
SELECT \*  
FROM project1.actors  
ORDER BY LAST\_NAME ASC  
        ,FIRST\_NAME ASC  
LIMIT 10;

DISTINCT

Let's say you want to know all of the ratings for the movies   
SELECT RATING  
FROM movies;

But don’t like the duplicate rows and want to only display unique ratings.  
SELECT DISTINCT RATING  
FROM movies;

Distinct only works when the whole row returned is unique. In the query below, because each movie has a unique ID, none of the rows are removed.  
SELECT DISTINCT MovieID  
               ,RATING  
FROM movies;

**COUNT , GROUP BY, LIKE**

Count can be used as follows:

SELECT COUNT(\*) AS cnt  
FROM movies;

This will return the number of movies in the table

Try the following to get the number of movies for each rating and you'll get incorrect results:  
SELECT RATING  
      ,COUNT(\*) AS cnt  
FROM movies;  
\*Incorrect Results

You can use GROUP BY (column name) to get correct results in such cases:  
SELECT RATING  
      ,COUNT(\*) AS MOVIES  
FROM movies  
GROUP BY RATING;

Like is used all the time for search when an entire description is not known, it's used with a % -> LIKE %  
SELECT \*  
FROM movies  
WHERE TITLE like 'Night at the Museum%';

Name the return columns with spaces using:  
SELECT COUNT(\*) AS "Night at the Museum Movies"  
FROM movies  
WHERE TITLE like 'Night at the Museum%';

You can put % before the word as well so like '%Museum%' and it will return all results that have museum in the title

**MIN & MAX**

SELECT MAX(ReleaseYear)

FROM movies;

They work exactly how you would expect.

SELECT Title  
      ,MAX(ReleaseYear)  
      ,RATING  
FROM movies;  
\*Incorrect results

SELECT MIN(ReleaseYear)  
      ,RATING  
FROM movies  
WHERE Title like 'Night at the Museum%'  
GROUP BY Title  
        ,RATING;

SELECT RATING  
      ,MIN(ReleaseYear)  
FROM movies  
WHERE Title like 'Night at the Museum%'  
GROUP BY RATING;

SELECT RATING  
      ,MIN(ReleaseYear)  
FROM movies  
GROUP BY RATING;

SELECT RATING  
      ,MAX(ReleaseYear)  
FROM movies  
GROUP BY RATING;

MySQL is more “flexible” than other SQL versions.  It doesn’t require the GROUP BY but it should.  It is likely you will get unhelpful or incorrect results when you do not use a group by.

All result sets should either be part of an aggregate function, or in the Group By.

**DATA TYPES – BOOLEAN & DECIMAL**

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 LIKE**

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.

**COMPARISION OPERATORS**

>, <, BETWEEN, IN, NOT IN, OR

Combination of both Greater Than and Less Than is <> which means Not Equal.

Just like = and <>, we can use IN to include and NOT IN to exclude multiple values at the same time

**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

**DATE & TIME** **FUNCTIONS:**

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();

**PRIMARY AND FOREIGN KEYS**

A primary key keeps every row unique.

Foreign key – Primary key in one table used as Foreign key in another table

If we join two tables together then Primary keys of respective tables become Foreign keys in this joined table.