**%load\_ext sql**

**%sql mysql://studentuser:studentpw@mysqlserver/dognitiondb**

**%sql USE dognitiondb**

**WEEK 1**

1. SQL: structured query language

Teradata: data warehouse

2. Goal:

1. Describe the structure of relational databases;

2. Interpret and create entity-relationship diagrams and relational schemas that describe the contents of specific databases;

3. Write queries that retrieve and sort data that meet specific criteria, and retrieve such data from MySQL and Teradata databases that contain over 1 million rows of data;

4. Execute practices that limit the impact of your queries on other coworkers;

5. Summarize rows of data using aggregate functions, and segment aggregations according to specified variables;

6. Combine and manipulate data from multiple tables across a database;

7. Retrieve records and compute calculations that are dependent on dynamic data features, and translate data analysis questions into SQL queries.

This course will provide you with a highly coveted skill set, making you a very attractive candidate to data analyst recruiters. You will also have a powerful set of tools to help provide real, tangible value to your business. Many opportunities will open up for you as you embrace this important feature of the data-driven business world.

3. Good data storage allows:

* Easy retrieval
* Easy updating
* Accessibility for multiple people at the same time
* Data consistency
* Space efficiency
* Speed
* Security

4. **Relational database**

MySQL, TERADATA

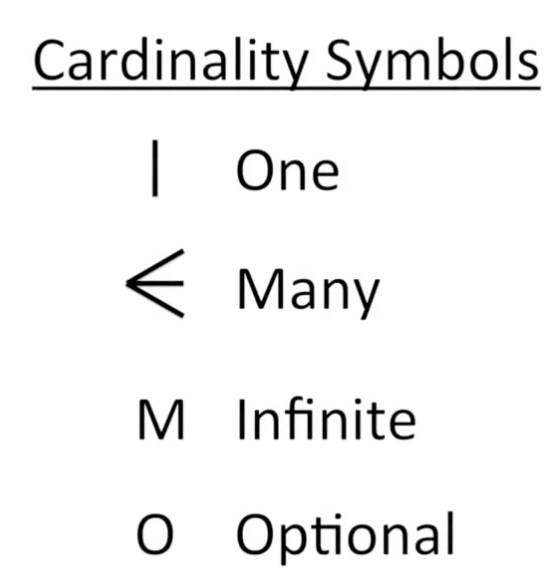
* Tables in relational database
* Link the tables
* Sets in algebra
* Tables=smallest logical subsets of data
* Each column represents a unique category of information
* Each row must be unique
* Order of columns or rows doesn’t matter

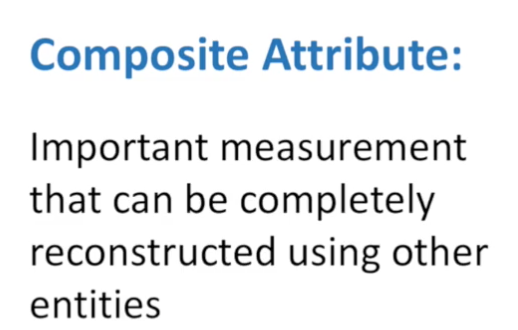
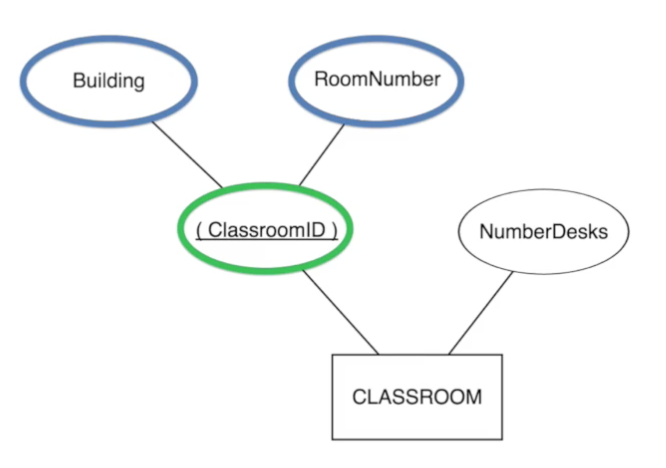
5. Database design tools

* Query
* Entity-Relationship diagrams (ER diagrams)
* Relational Schemas

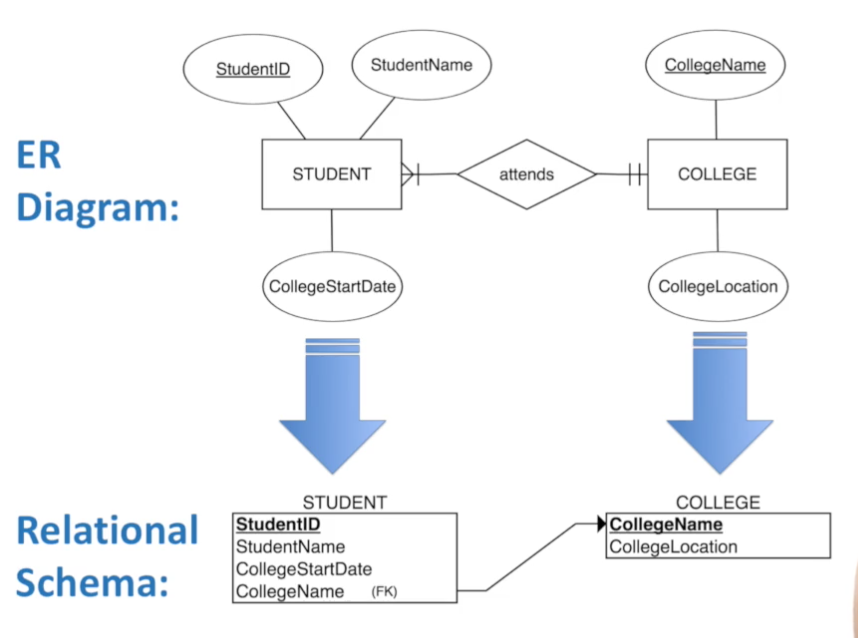
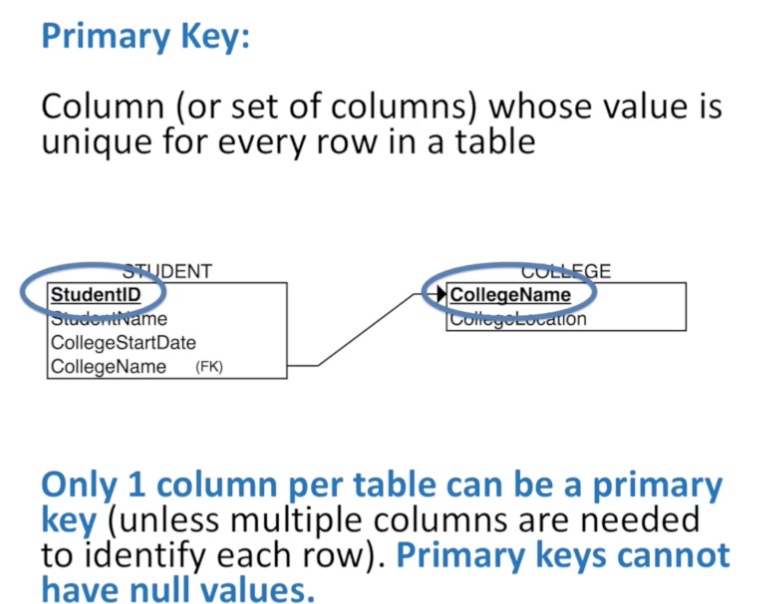
6. ER diagrams

* Theme
* Boxes: Entities: categories of similar, but unique measurements
* Ovals: Attributes: unique measurements within a category
* A attributes must be connected to at least one entities
* Entity instance: rows of each table
* Unique attribute: an attribute with a unique value in each entity instance, e.g., the column that allows you to link tables together
* Diamond (菱形): relationship
* Along the reading direction: minimum then maximum;
* Cardinality Symbols: |: one, M: Infinite
* Specific cardinality constrain notation: (100,1000) Numbers take precedence over symbols; Numbers are always written from left to right (minimum to maximum)



* Composite Attribute: ClassroomID
*  

7. Relational Schemas

* Key Components: Tables, Primary Keys, Foreign Keys
* 
* Vocabulary Terms: relation (is like, but not equivalent to table)
* Table (technical term: relation)
* Column, Field (technical term: attribute): Theory/Practice: no order; Each column in a table should represent a unique category of information
* Row (technical term: tuple): Theory: tuple cannot be duplicated; Practice: rows can be duplicated. Theory/Practice: no order
* **Primary Keys:**
* ****
* **Foreign Key: Column that refer to another primary key of another table**
* 

8. ERDPlus: Make ER diagrams

9. ERDPlus: Make Relational Schemas

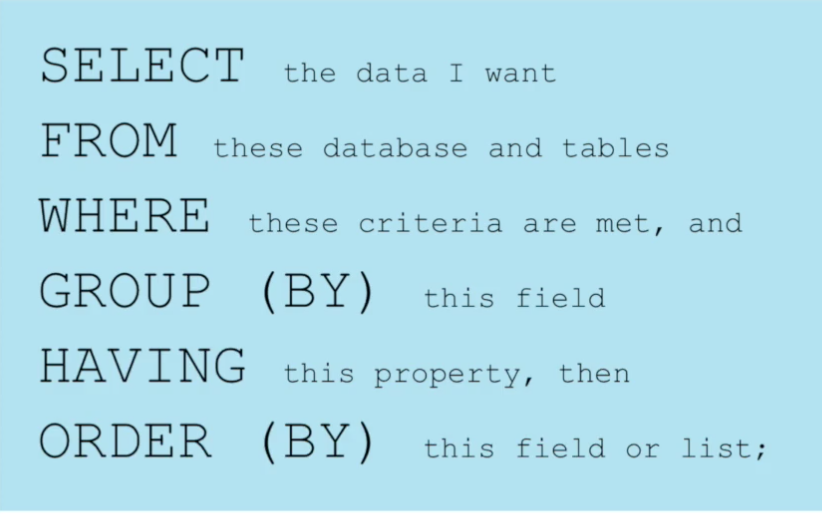
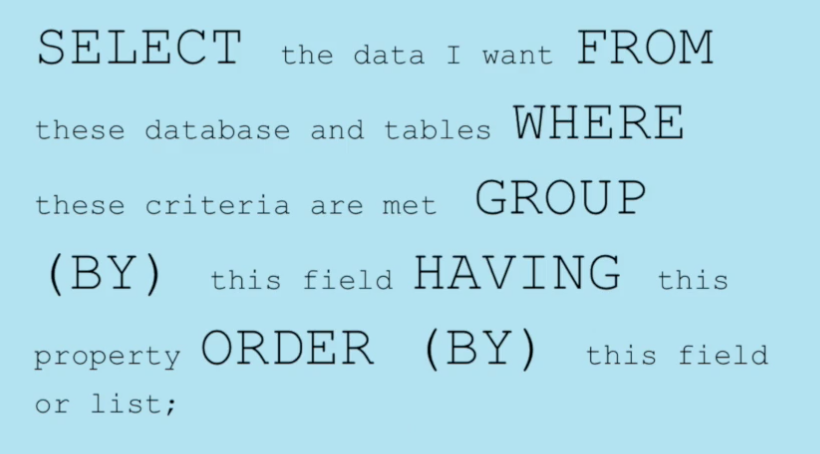
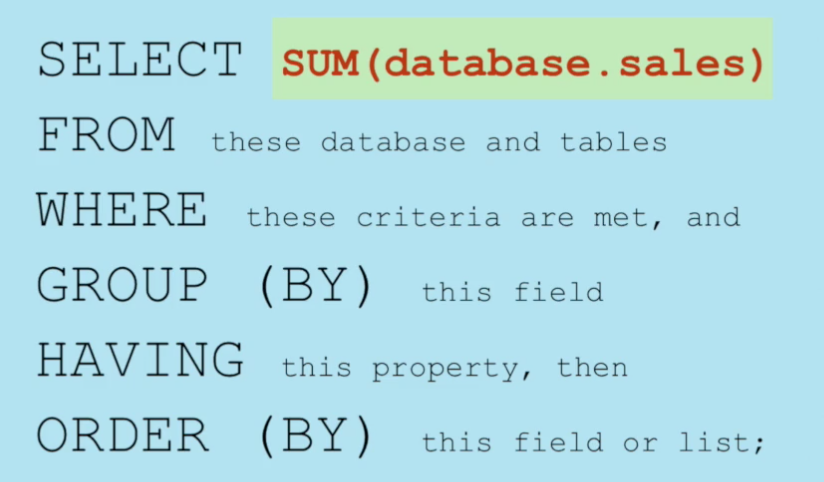
**WEEK 2**

1. Dognition Data

2. Dillard’s Data

* items
* How much they cost
* Date of purchase
* Store location
* Census information from store location

3. Introduction to query syntax

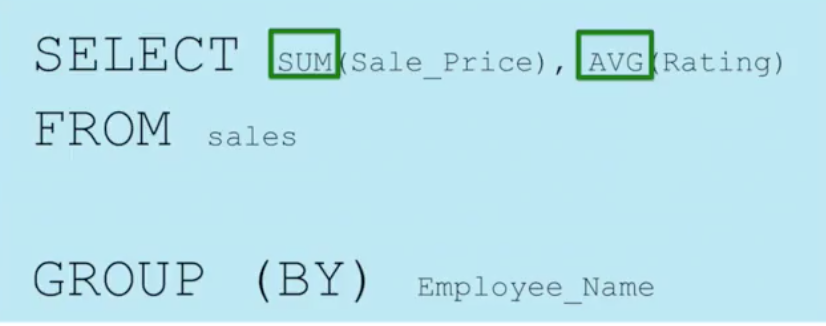
* Data Query Language (DQL): retrieve data
* Query: SQL code that describes the data your desire and the format in which you want it
* Front end interface: where users enter queries
* **SELECT, FROM, WHERE, GROUP(BY), HAVING, ORDER (BY);**
* (RECOMMEND THIS, good for review)
* 
* function: SUM
* 
* No data = NULL (data is missing)

4. Teradata Viewpoint and SQL Scratchpad

* Your Teradata login is: **DUKESQLMOOC7822**
* Your Teradata password is: **XG88fr$**
* Teradata use **TOP instead of** LIMIT

**WEEK 3**

1. Aggregate functions

* Count, SUM, AVG, MAX
* 
* Every row of output from a database query has to use the same aggregation “level”
* When a column is included in a count function, null values are ignored in the count. When an asterisk is included in a count function, nulls are included in the count.
* You can use the "\*" in the parentheses of a COUNT function to count how many rows are in the entire table (or subtable). There are two fundamental difference between COUNT(\*) and COUNT(column\_name), though. **The first difference is that you cannot use DISTINCT with COUNT(\*).**

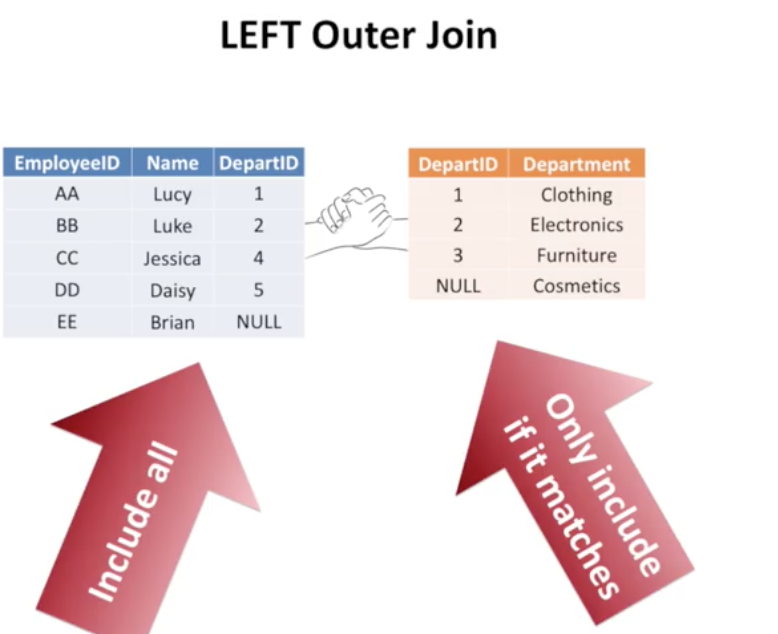
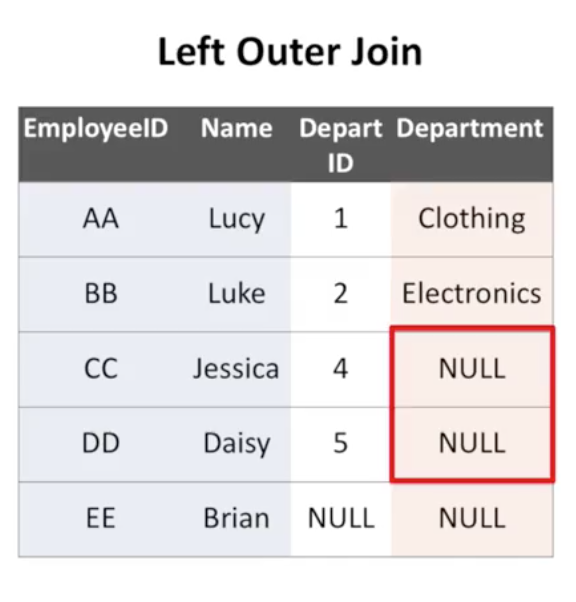
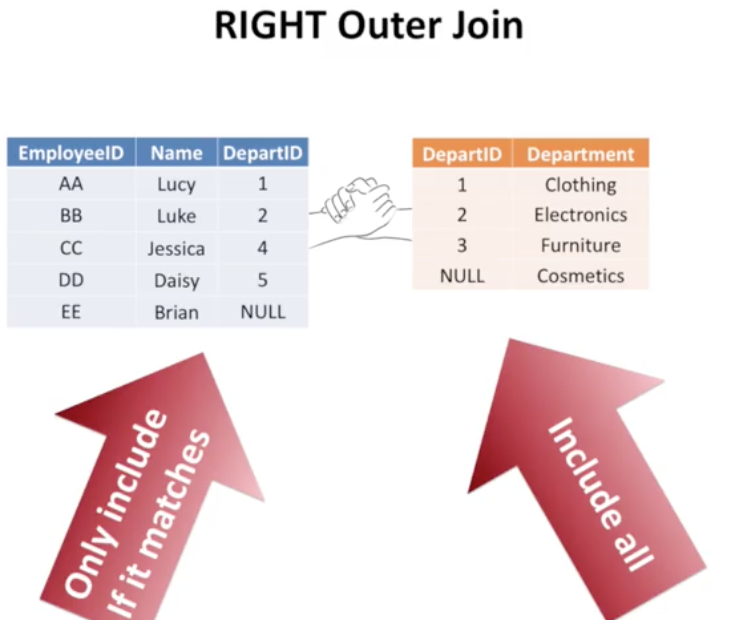
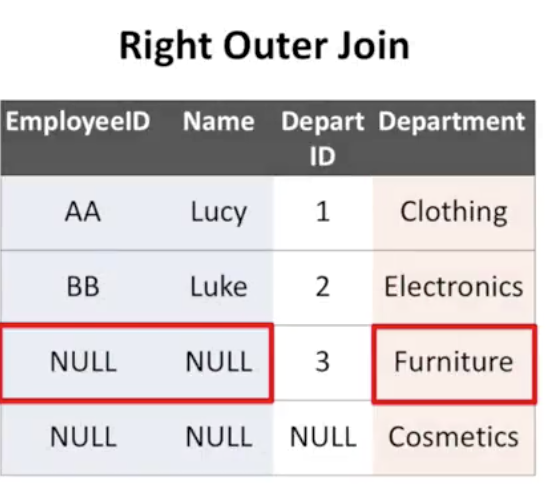
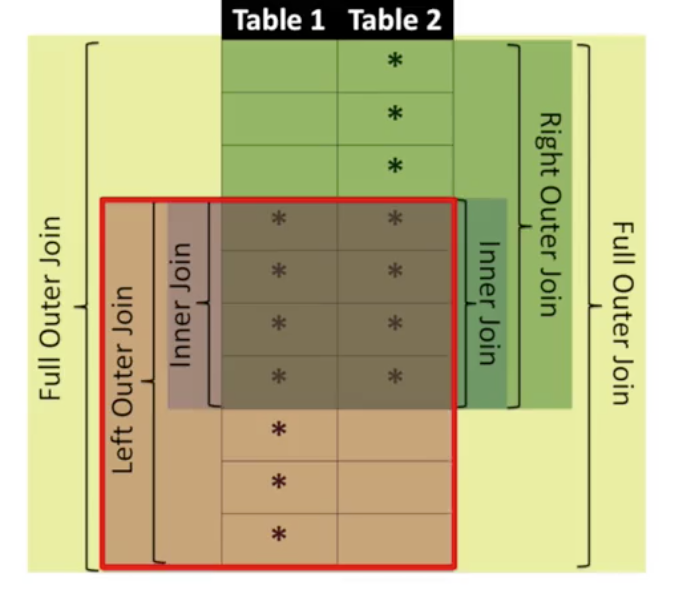
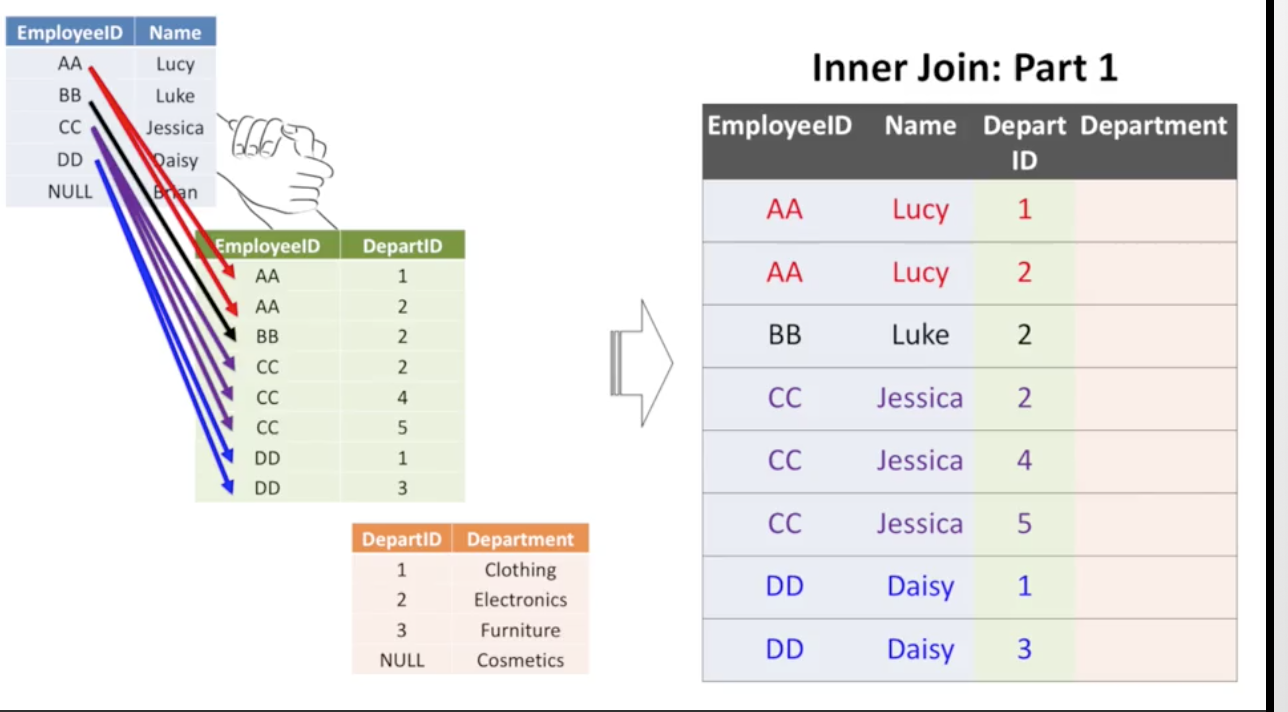
**SUM**

* **SELECT** **SUM**(ISNULL(exclude))
* **FROM** dogs
* Detect how many NULL in column ‘exclude’

**HAVING clause**

HAVING clause has to be applicable or computable using a group of data.

**JOINS**

* Set theory: Cross product of sets
* Inner Join (“NULL” value cannot be matched)
* Outer Join : Left outer join, right outer join, full outer join
* **** ****
* **** ****
* ****
* **** ****
* **MySQL does not support full outer join!**
* ****
* **Inner join is not always smaller than the input sets**
* ****
* **The order of joins is matter!**
* **Duplicates: clean data and avoid duplicates as many as possible**

**Join Syntax:**

I find this syntax -- called the "equijoin" syntax -- to be very intuitive, so I thought it would be a good idea to start with it. However, we can re-write the inner joins in the same syntax used by outer joins. To use this more traditional syntax, you have to tell the database how to connect the tables using an ON clause that comes right after the FROM clause. Make sure to specify the word "JOIN" explicitly. This traditional version of the syntax frees up the WHERE clause for other things you might want to include in your query. Here's what one of our queries from the inner join lesson would look like using the traditional syntax:

**SELECT** d.dog\_guid **AS** DogID, d.user\_guid **AS** UserID, **AVG**(r.rating) **AS** AvgRating, **COUNT**(r.rating) **AS** NumRatings, d.breed, d.breed\_group, d.breed\_type

**FROM** dogs d **JOIN** reviews r

**ON** d.dog\_guid=r.dog\_guid **AND** d.user\_guid=r.user\_guid

**GROUP** **BY** d.user\_guid

**HAVING** NumRatings > 9

**ORDER** **BY** AvgRating **DESC**

**LIMIT** 200

You could also write "INNER JOIN" instead of "JOIN" but the default in MySQL is that JOIN will mean inner join, so including the word "INNER" is optional.

If you need a WHERE clause in the query above, it would go after the ON clause and before the GROUP BY clause.

When we use the traditional join syntax to write inner joins, the order you enter the tables in your query doesn't matter. In outer joins, however, the order matters a lot. A left outer join will include all of the rows of the table to the left of the LEFT JOIN clause. A right outer join will include all of the rows of the table to the right of the RIGHT JOIN clause. So in order to retrieve a full list of dogs who completed at least 10 tests in the reviews table, and include as much breed information as possible, we could query:

**SELECT** r.dog\_guid **AS** rDogID, d.dog\_guid **AS** dDogID, r.user\_guid **AS** rUserID, d.user\_guid **AS** dUserID, **AVG**(r.rating) **AS** AvgRating, **COUNT**(r.rating) **AS** NumRatings, d.breed, d.breed\_group, d.breed\_type

**FROM reviews r LEFT JOIN dogs d**

**ON** r.dog\_guid=d.dog\_guid **AND** r.user\_guid=d.user\_guid

**WHERE** r.dog\_guid **IS** **NOT** NULL

**GROUP** **BY** r.dog\_guid

**HAVING** NumRatings >= 10

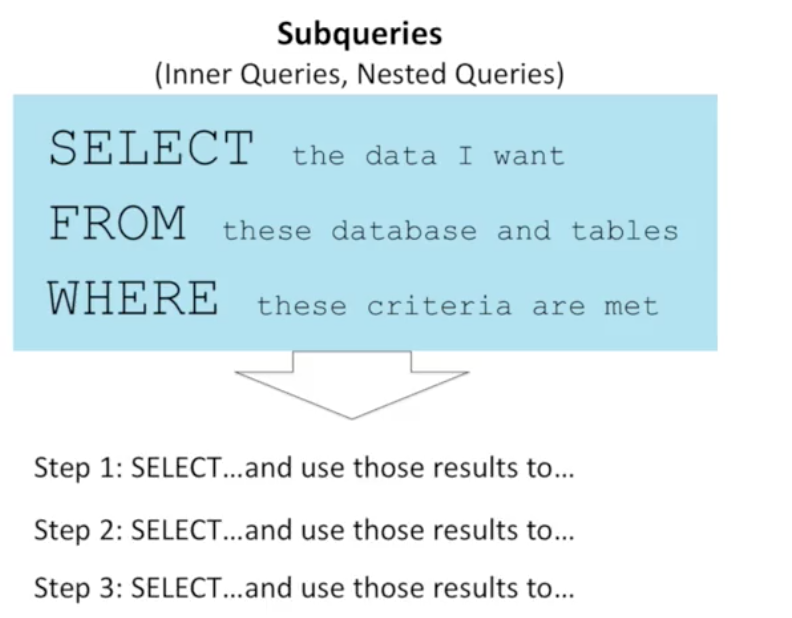
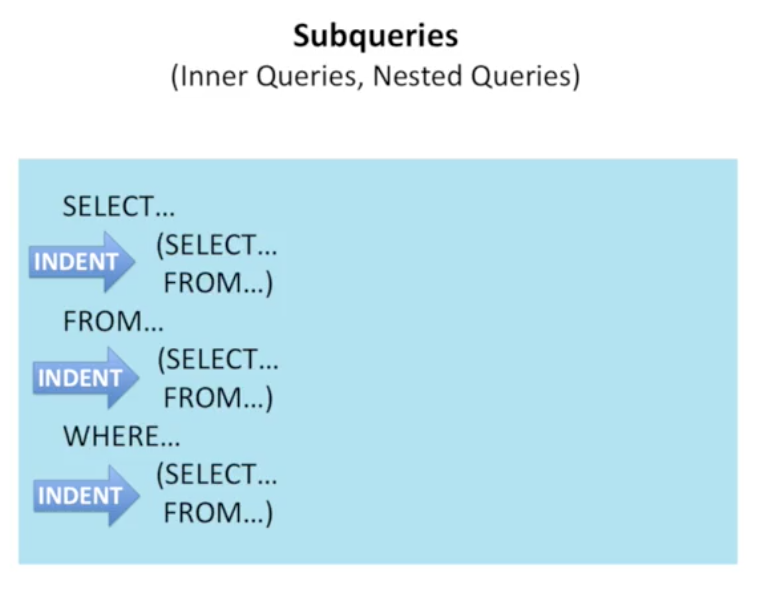
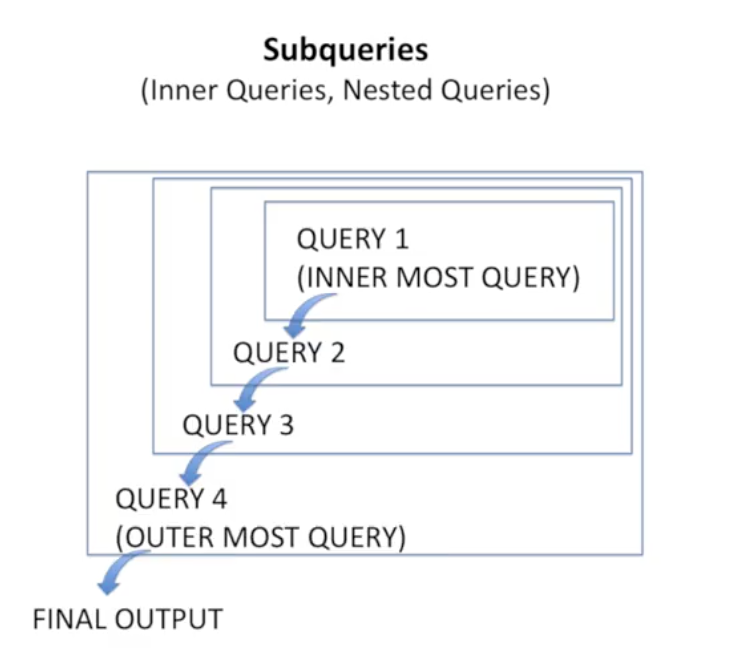
**ORDER** **BY** AvgRating **DESC**;

That's because COUNT DISTINCT does NOT count NULL values, while SELECT/GROUP BY clauses roll up NULL values into one group. If you want to infer the number of distinct entries from the results of a query using joins and GROUP BY clauses, remember to include an "IS NOT NULL" clause to ensure you are not counting NULL values.

* Avoid making assumptions about your data or your analyses. For example, rather than assume that all the values in a column are unique just because some documentation says they should be, check for yourself!
* Always look at example outputs of your queries before you strongly interpret aggregate calculations. Take extra care to do this when your queries require joins.
* When your queries require multiple layers of functions or joins, examine the output of each layer or join first before you combine them all together.
* Adopt a healthy skepticsm of all your data and results. If you see something you don't expect, make sure you explore it before interpreting it strongly or incorporating it into other analyses.

**Week 4**

1. Subqueries

* 
* 
* 

**Analysis Plan**

**Logical expression**

IF([your conditions],[value outputted if conditions are met],[value outputted if conditions are NOT met])

So we could write:

**SELECT** created\_at, IF(created\_at<'2014-06-01','early\_user','late\_user') **AS** user\_type

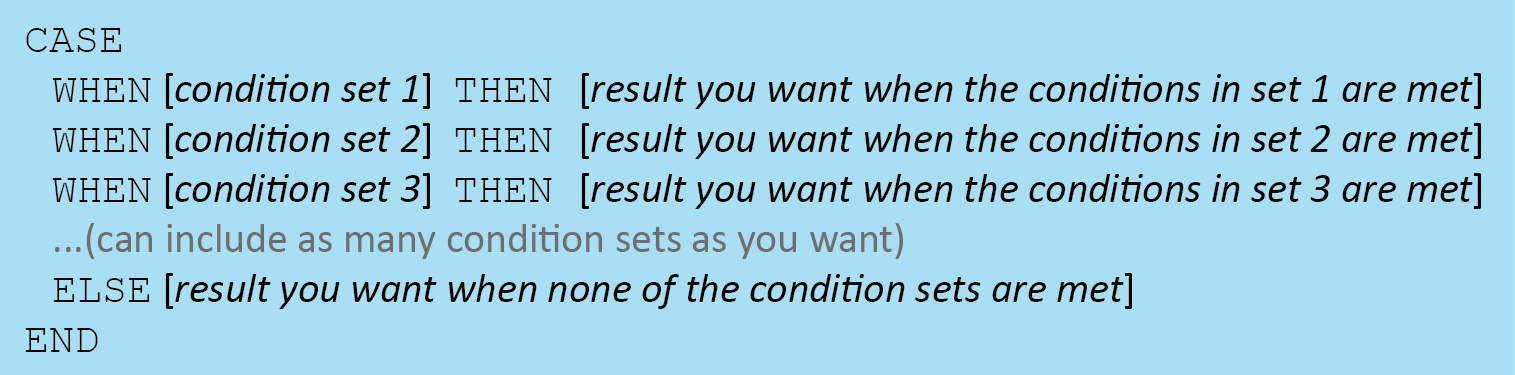
**FROM** users

When you nest IF expressions, it is important to encase each IF expression--as well as the entire IF expression put together--in parentheses.

The IF function is not supported by all database platforms, and some spell the function as IIF rather than IF, so be sure to double-check how the function works in the platform you are using.

**CASE**

The main purpose of CASE expressions is to return a singular value based on one or more conditional tests. You can think of CASE expressions as an efficient way to write a set of IF and ELSEIF statements. There are two viable syntaxes for CASE expressions. If you need to manipulate values in a current column of your data, you would use this syntax:



Using this syntax, our nested IF statement from above could be written as:

**SELECT** CASE WHEN cleaned\_users.country="US" THEN "In US"

WHEN cleaned\_users.country="N/A" THEN "Not Applicable"

ELSE "Outside US"

END **AS** US\_user,

**count**(cleaned\_users.user\_guid)

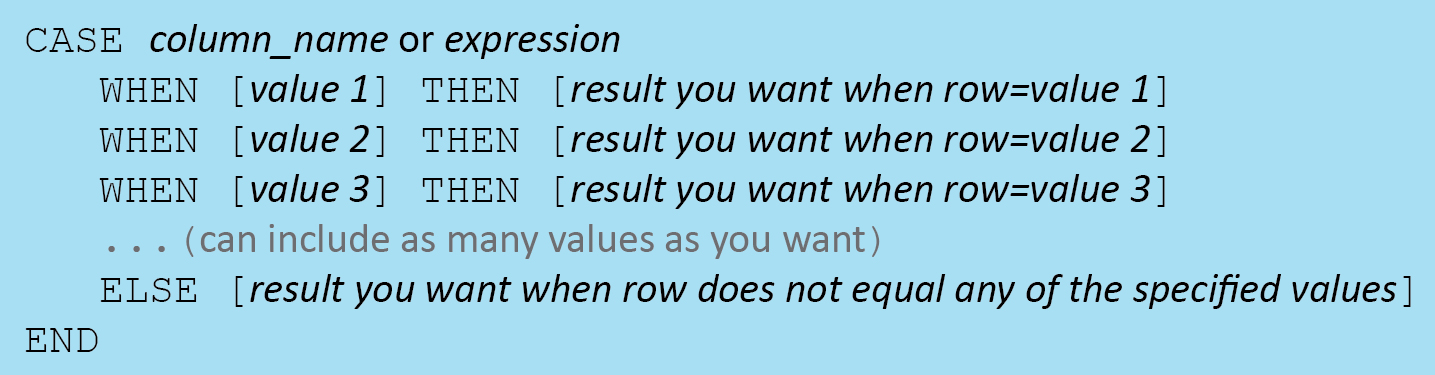
**FROM** (**SELECT** **DISTINCT** user\_guid, country

**FROM** users

**WHERE** country **IS** **NOT** NULL) **AS** cleaned\_users

**GROUP** **BY** US\_user

Since our query does not require manipulation of any of the values in the country column, though, we could also take advantage of this syntax, which is slightly more compact:



Our query written in this syntax would look like this:

**SELECT** CASE cleaned\_users.country

WHEN "US" THEN "In US"

WHEN "N/A" THEN "Not Applicable"

ELSE "Outside US"

END **AS** US\_user,

**count**(cleaned\_users.user\_guid)

**FROM** (**SELECT** **DISTINCT** user\_guid, country

**FROM** users

**WHERE** country **IS** **NOT** NULL) **AS** cleaned\_users

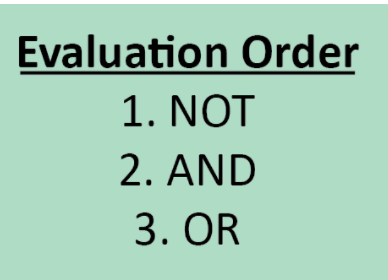
**GROUP** **BY** US\_user

There are a couple of things to know about CASE expressions:

* Make sure to include the word END at the end of the expression
* CASE expressions do not require parentheses
* ELSE expressions are optional
* If an ELSE expression is omitted, NULL values will be outputted for all rows that do not meet any of the conditions stated explicitly in the expression
* CASE expressions can be used anywhere in a SQL statement, including in GROUP BY, HAVING, and ORDER BY clauses or the SELECT column list.

**Cannot use 1<weight<=10, BUT using 1<weight AND weight <=10**

**unless parentheses are included, the NOT operator is always evaluated before an AND operator, and an AND operator is always evaluated before the OR operator.**



**Equijoin syntax:**

%%sql

SELECT d.dog\_guid AS DogID, d.dimension AS dimension, COUNT(c.created\_at)AS Num

FROM dogs d, complete\_tests c

WHERE d.dog\_guid=c.dog\_guid

GROUP BY DogID

LIMIT 100;

QUIZ 5

1. YES

2. 13623

3. Atlanta, GA

4. 3949538

5. 5715232

6. Hart Sch

7. Metairie, LA

8. MIN(MEDIAN) 2707, MAX 3902 McAllen TX WRONG

9. low

10. 25452

11. Pine Bluff, AR WRONG

12. Louisville, KY

13. Clinique, KY

14. Jan and Aug

15. Des