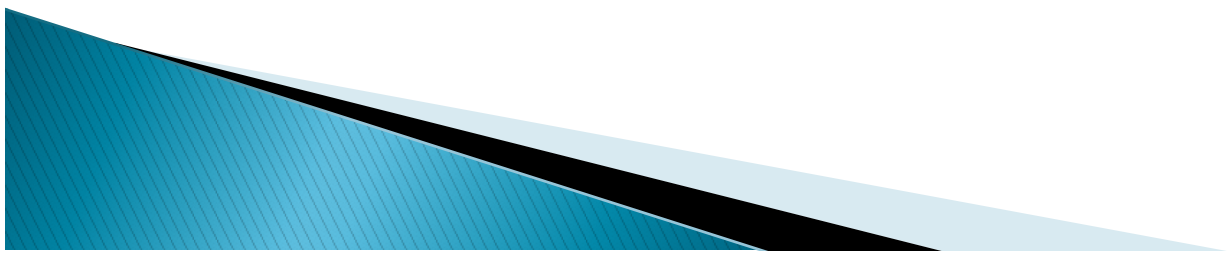
CSCI-UA.60-1 Database Design and Web Implementation

MySQL – Part 2

Prof Deena Engel
Department of Computer Science
deena.engel@nyu.edu

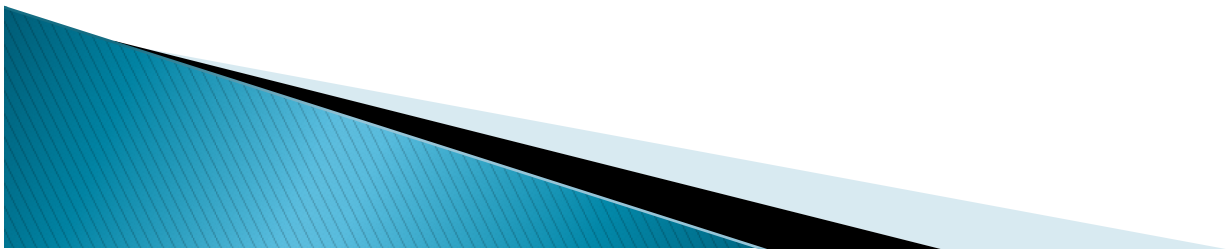
Class Notes

- Please refer to the transcripts and class notes that are posted to the course home pages for specific programming examples!



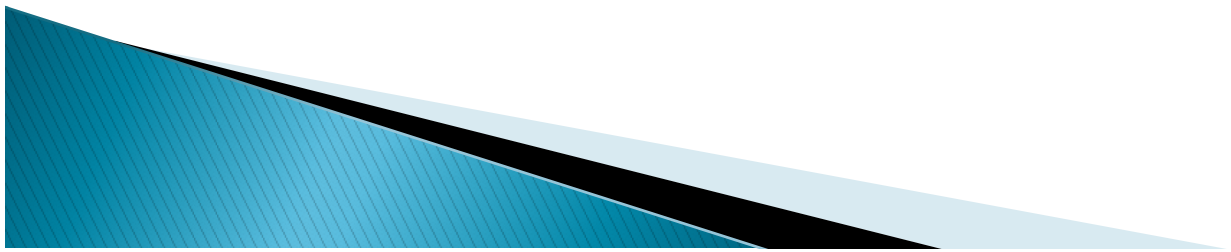
MySQL: Joining tables

- In this unit of the course, we will discuss JOINS and additional topics relating to *working with more than one table at a time*.
- Joins in SQL are used to "match up" tables using the relationships we have discussed with respect to database design.
- There are many types of Joins in MySQL: INNER JOIN, LEFT JOIN, RIGHT JOIN commands as well as *reflexive* joins.
- We will also discuss *implicit* and *explicit* joins.
- When we complete this study, we will discuss other ways of combining data from different tables such as UNION.
- We will also examine *sub-queries* as a way to combine queries within a single statement when needed.



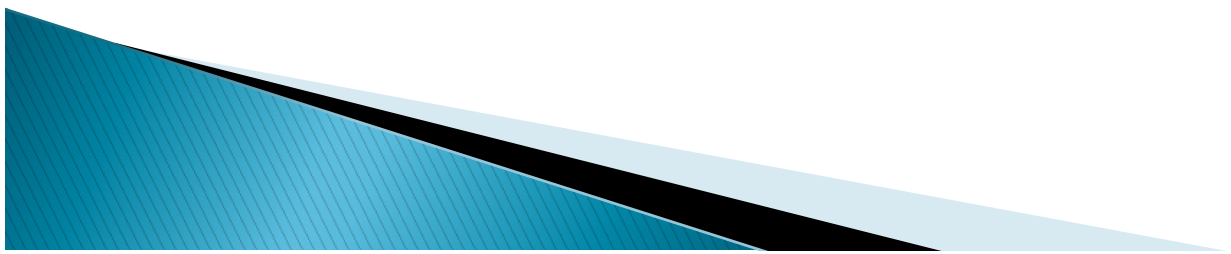
Using INNER joins: implicit and explicit joins

- We will be begin with INNER joins which can typically be written in two ways:
  - Implicit join
  - Explicit join
- When we join two tables with an INNER join in a SELECT statement, only records which have a match in both tables will be selected.
- Most of our discussion to date with respect to relational database design has focused on the concept of the INNER JOIN.
- By definition … INNER JOINs comply with *referential integrity*.



LEFT and RIGHT joins

- LEFT joins allow the programmer to see all of the results in the source table … even if there is no match in the related table.
- A LEFT join that brings up unmatched results does not necessarily signify problems with the data.



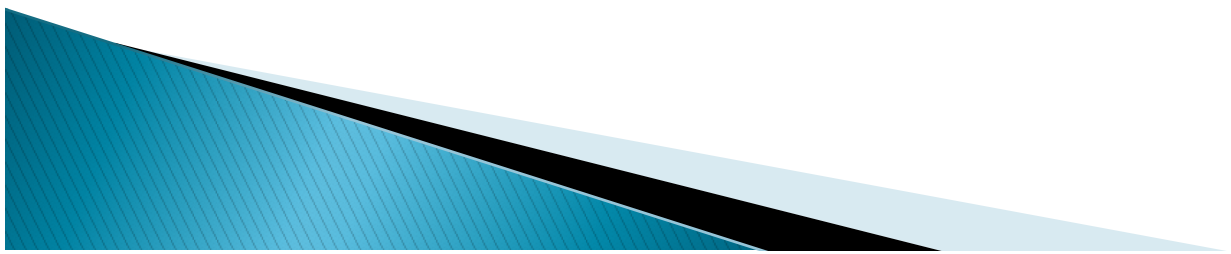
An example of analysing the results of a LEFT join:

For example … If one were to envision a table of all undergraduate students related to one or more tables of grades and transcript data by "N-#" where the "N-#" is the primary key in the students' table and the foreign key in the transcripts and grades tables:

- Once incoming freshman enroll, they are assigned an "N-#" in the summer before they start classes. If one were to query all undergraduates in the month of September, one would find many students who have enrolled … but who do not have records in the transcripts table(s). In this case, those students are the newly enrolled freshman and transfer students who have not yet received any official grades and there is no indication of a problem or anomaly.
- However, students who do not have a transcript after mid-semester grades have been posted … are either students who enrolled but are not attending (whom the Registrar would like to know about!) … or perhaps there are one or more instructors who have not posted mid-semester grades (also a problem the Registrar would like to know about!)

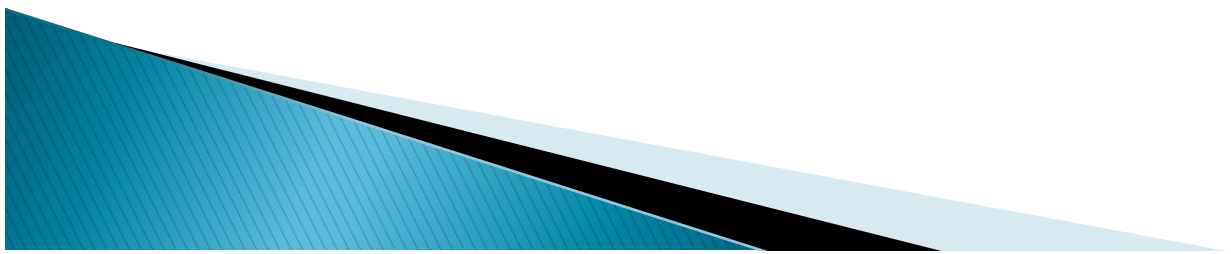
RIGHT joins

- RIGHT joins will bring up records in the related table regardless of whether there is a match.
- In our example above: if “Albert” contains a transcript for which there is no student record … that is indicative of a serious data problem.



RIGHT joins, continued

- RIGHT joins are widely used when data tables that are part of a relationship are imported into large systems in order to determine the quantity and nature of problems.
- These problems are then addressed either programmatically as we have done in class earlier by “scrubbing” the data and then re-importing it into the database; or by writing SQL queries (or scripts that hold the queries) to “scrub” the data when possible.



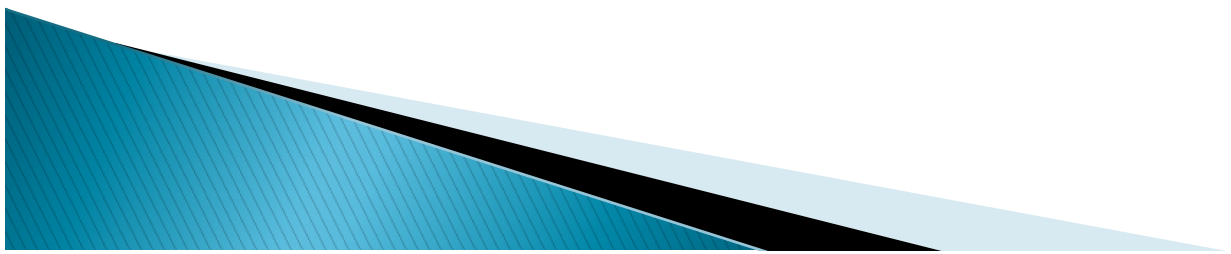
RIGHT JOINS and Referential Integrity

- If you need to suppress referential integrity while working with your data during a particular session:

```
SET foreign_key_checks = 0;
```

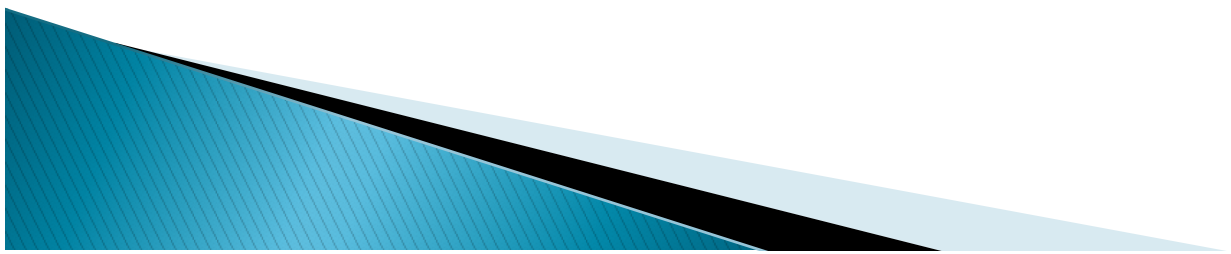
- Upon completion, you can turn it back on:

```
SET foreign_key_checks = 1;
```



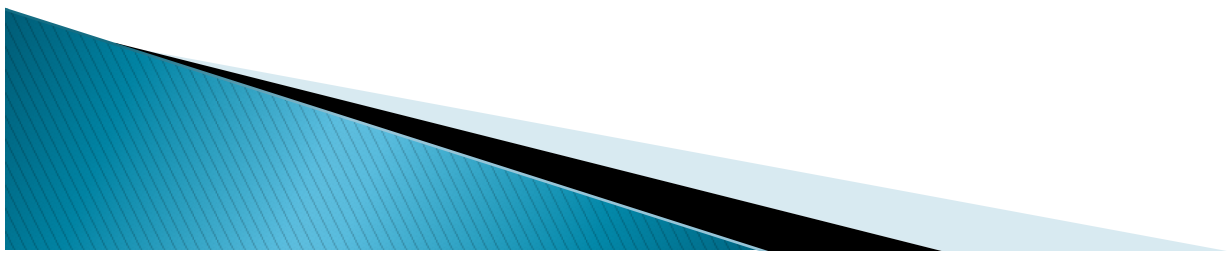
Union in MySQL

- The UNION statement is not a join operator.
- The UNION statement uses tables that have the same or similar structure and then allows you to combine them in order to work with a consolidated data set in a single query.



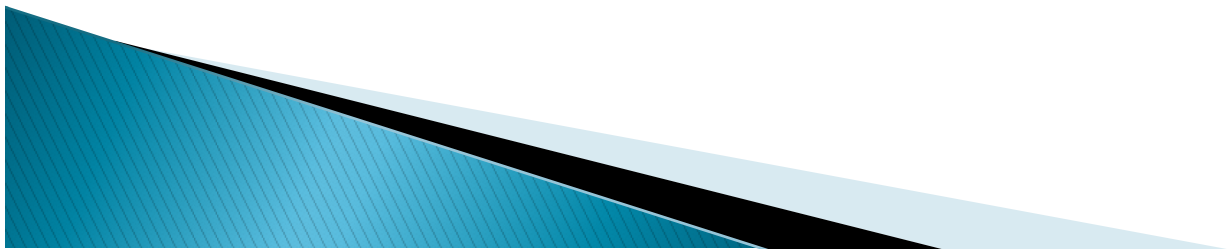
UNION: case study example

- UNION is especially useful in situations where there are many data files to work from. For example, a financial institution might store daily transactions on a data server in files or tables with the same structure, each containing data for one day.
- One can then use UNION to run statistical analysis on the past 3 days for example … or the past 5 Fridays … or other combinations.
- Of course, one can use the WHERE clause on a table holding all of the data to select data by date. However, in the case described here, a single file would become very large, very quickly and would require a great deal of work to maintain with respect to archiving needs, etc. So for research purposes (not for transaction processing), one could use MySQL's UNION to consolidate the data that are needed for a given SELECT query.



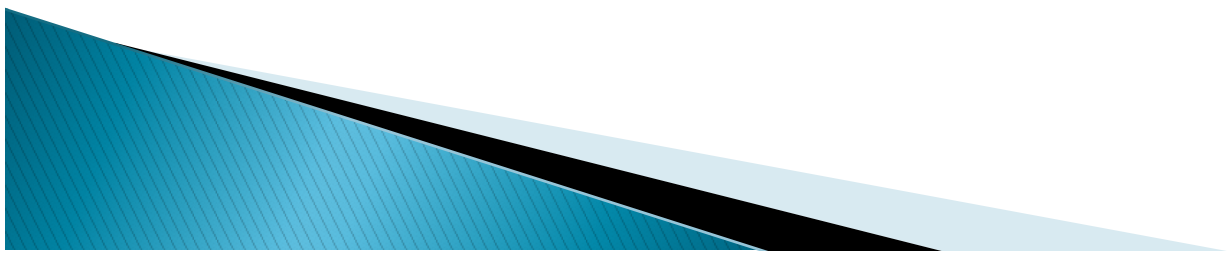
Reflexive-Joins (also called "Self-Joins")

- There are situations in which one would want to build a join within the same table. This can only work if there is one relationship:
  - For example, in a table of employees … one might notate who the manager is for each employee and the managers in turn each have a manager of their own
  - For example, in a table of clients who might jointly own accounts, each spouse is spouse to the other person.



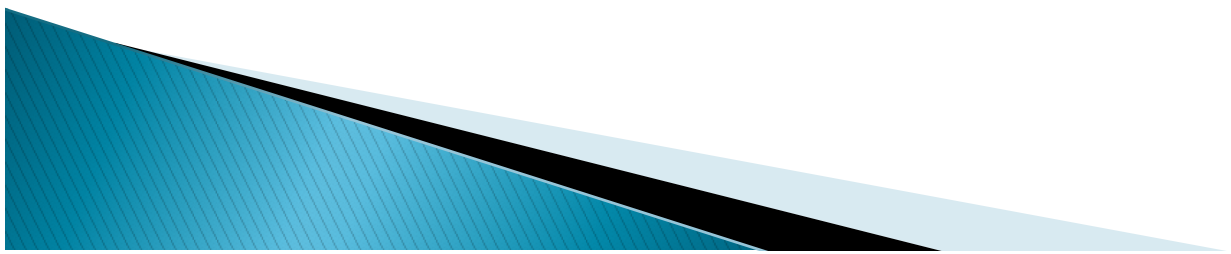
Be wary of reflexive joins …

- … because you might in fact need a MANY-TO-MANY relationship. In the examples on the previous slide:
  - An employee can have many managers over time or even report to different managers simultaneously for different projects
  - Joint-ownership of an account can change over time.



User Variables and the Assignment Operator in MySQL

- MySQL offers the option to assign a single data result to a variable.
- This can be quite useful when building a user-interface!
- The assignment operator as we have discussed in class is :=
- You can use the variable in the following MySQL statement if needed.



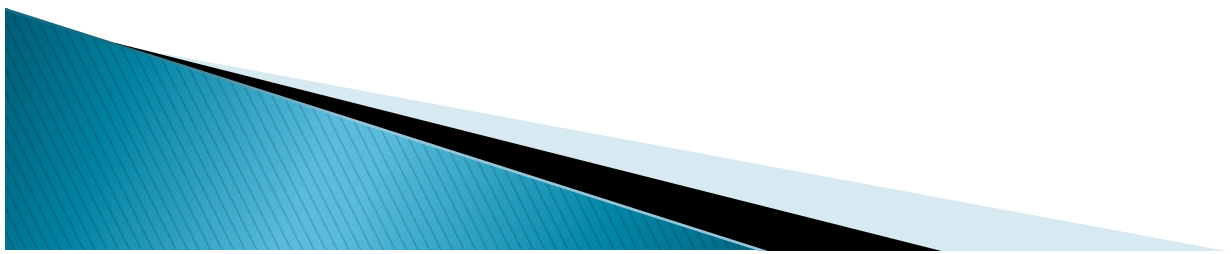
An example using a variable:

```
mysql> SELECT @avgPrice := AVG(appraisal_amount) FROM paintings;
+-------------------------------------+
| @avgPrice := AVG(appraisal_amount) |
+-------------------------------------+
|                  9650000.000000000 |
+-------------------------------------+
1 row in set (0.01 sec)

mysql> SELECT artist_name,title,appraisal_amount
    -> FROM paintings
    -> WHERE appraisal_amount > @avgPrice
    -> ORDER BY appraisal_amount DESC;
+-------------------+--------------------------+------------------+
| artist_name       | title                    | appraisal_amount |
+-------------------+--------------------------+------------------+
| Van Gogh, Vincent | The Starry Night         |      38000000.00 |
| da Vinci, Leonardo | Last Supper             |      30000000.00 |
| Vermeer, Johannes | Girl with a Pearl Earring |      25000000.00 |
| Picasso, Pablo    | Les Demoiselles d'Avignon |      13000000.00 |
| Rembrandt van Rijn | Self-Portrait           |      10000000.00 |
+-------------------+--------------------------+------------------+
5 rows in set (0.01 sec)
```

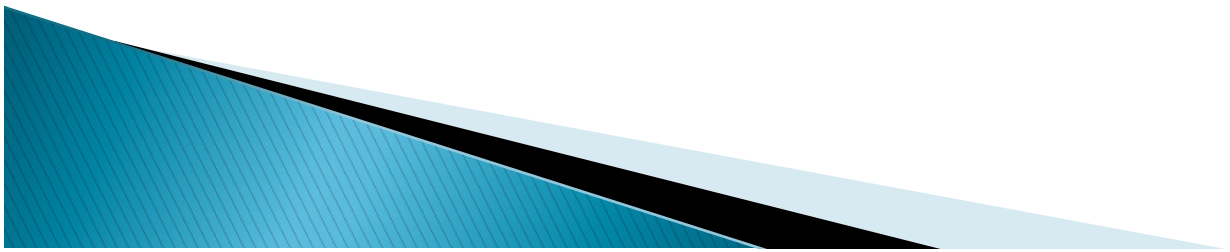
Sub-queries … or "Nested queries"

- If you find yourself using variables, it is a good idea to try to combine the statements using a sub-query.
- A sub-query must be a valid query in its own right; and in fact, many programmers write the sub-query first in order to be sure that it returns the expected results.
- Notice that the sub-query is enclosed in parentheses but does not require a semi-colon at the end.



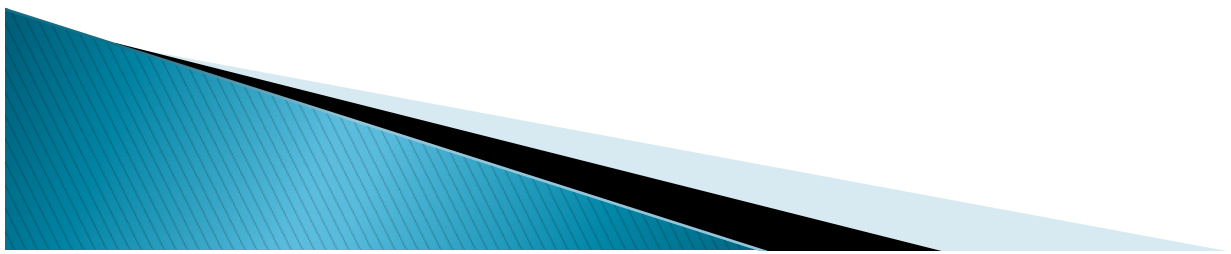
Sub-query example

```
mysql> SELECT artist_name,title,appraisal_amount
        FROM paintings
        WHERE appraisal_amount >
           (SELECT AVG(appraisal_amount) FROM paintings)
        ORDER BY appraisal_amount DESC;
+--------------------+---------------------------+------------------+
| artist_name        | title                     | appraisal_amount |
+--------------------+---------------------------+------------------+
| Van Gogh, Vincent  | The Starry Night          |      38000000.00 |
| da Vinci, Leonardo | Last Supper               |      30000000.00 |
| Vermeer, Johannes  | Girl with a Pearl Earring |      25000000.00 |
| Picasso, Pablo     | Les Demoiselles d'Avignon |      13000000.00 |
| Rembrandt van Rijn | Self-Portrait             |      10000000.00 |
+--------------------+---------------------------+------------------+
5 rows in set (0.00 sec)
```



Use of IN, NOT IN

- MySQL supports IN and NOT IN with sub-queries. In general, one could use a JOIN instead.
- However, as we have discussed, it is important to know how to write the same query in different ways as there will be situations in which you wish to verify the results.

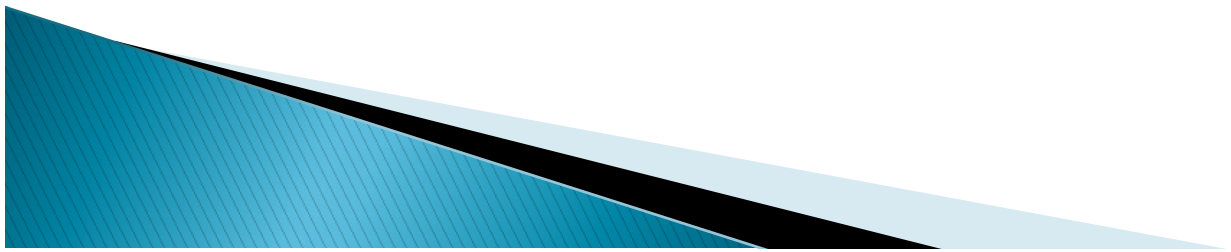


Example of IN

- Given the following example using IN with a sub-query … how would you re-write this with a JOIN and which JOIN would you use?

```
mysql> SELECT title,artist_name
    -> FROM paintings
    -> WHERE artist_name IN
    ->   (SELECT artist_name FROM artists WHERE origin = "France")
    -> ORDER BY artist_name,title;
```

- How might you re-write this to find all of the paintings by artists who are *not* French?

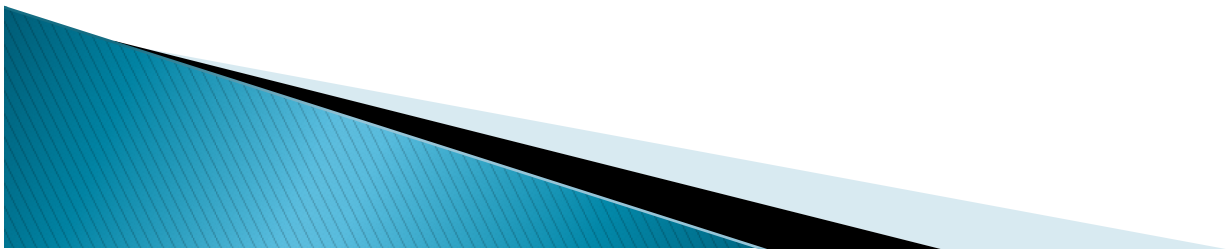


CREATE TABLE and SELECT

- One can build new tables out of SELECT queries.

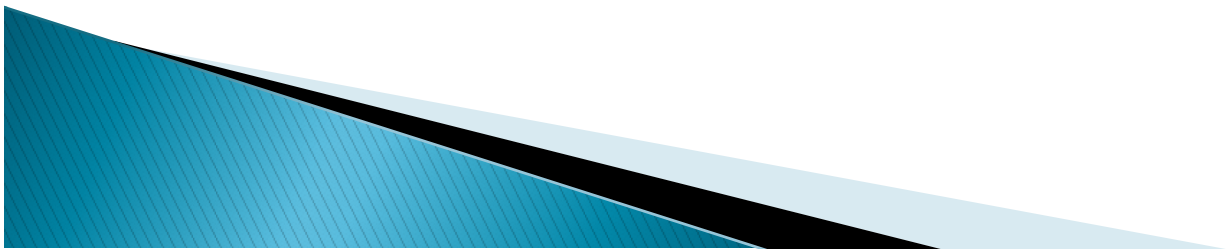
```
mysql> CREATE TABLE tmp_states AS
         (SELECT * FROM easternStates);
```

- In general, this not not recommended.
  - *For class discussion: Why not??*
- One could for example use the UNION command to create a table that can be easily exported.
  - *For class discussion:* Can you imagine situations in which this would be required?



De-Bugging Strategies

- Working with MySQL scripts and creating a database using a Query Language is quite different from writing programs in a programming language. Here are some suggestions to keep in mind.



DeBugging Strategies

- When you create and populate tables, organize the tables in the following order:
  - First, create and populate the tables that do not have any dependencies (e.g. reference tables)
  - Second create and populate the tables that have only one dependency.
  - Third continue in this fashion creating tables in an order which satisfies the condition that referential integrity is preserved throughout the process.
- In other words, one cannot create a foreign key … unless the primary keys that it matches to has already been created!

De-Bugging Strategies, continued

- It is helpful to break down a script by using comments (/* ….. */) to create and populate only one or a few tables at a time.
- You can run a script from within MySQL by using SOURCE … followed by SHOW WARNINGS to get more information about the errors.
  - This is analogous to using `.read` in SQLite.
- Error messages often capture an error from the line above due to the nature of the queries. So if you don't see an error on the indicated line, check the line above.

De-Bugging Strategies, continued

- Database implementations are greatly impacted by the content of the database. Do keep in mind during the debugging process whether your data and structures “pass the common sense test”. Sometimes bugs are due to anomalies in the data rather than in the SQL queries.