# EPBI 414

## Unit 6
## Basic Structured Query Language (SQL)

# The Recap - Unit 5

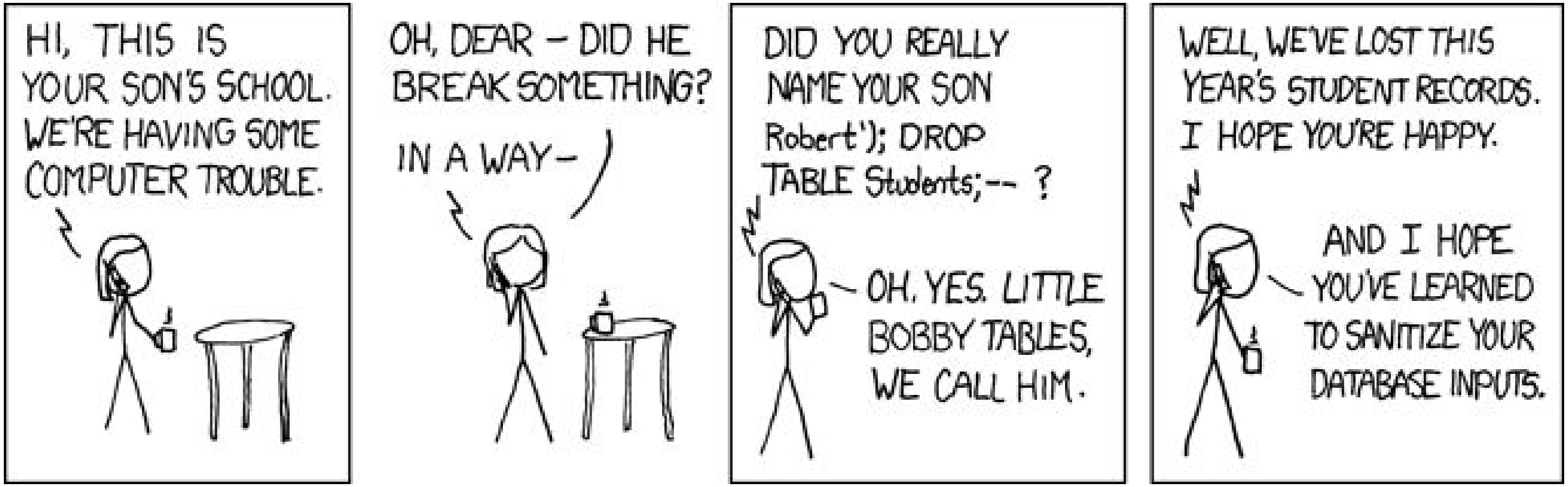
- The Unix / Linux operating system
  - The kernel and computing abstraction

- Working on the command line
  - Common *nix commands
  - File permissions and access
  - Scripting, pipes and redirects, input and output

# Unit 6 Overview

- Introducing SQL and MySQL
- Coding conventions / styles
- Connecting to MySQL
  - Command line, MySQL Workbench
- Basic SQL Programming
  - Queries, filtering, basic joins
  - Symbolic / programming logic

# Mandatory XKCD Comic


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

If nothing else, you should be able to understand this comic by the end of the SQL units.

# Introducing SQL

- Structured Query Language, or SQL (pronounced "sequel")

- Language designed for the relational model

- Many benefits:
    - Relative simple (small amount of keywords), but powerful

    - Open format (not a proprietary language)

# SQL vs. "SQL"

- The core of SQL is known as ANSI SQL
  - Most RDBMSs "more or less" adhere
- Pretty much every RDBMS makes their own version of SQL
  - Transact-SQL, MySQL (language), PL/SQL, PL/pgSQL
  - Generally proprietary, not compatible across systems

# For class

- We will work with MySQL in class, but there are many options
    - PostgreSQL, Oracle, SQLite, Sybase, Microsoft SQL Server
- However, we will stick to ANSI SQL to the maximum extent possible
    - You should do this anyway
    - Think carefully about whether you want to use the language-specific features

# MySQL

- One of the more popular open-source databases
    - Named after My, daughter of Michael Widenius (one of the original devs)

- Owned by Oracle, but available both open-source and enterprise (closed source)

- Was forked by Widenius into ***MariaDB*** (another daughter) to ensure openness

# Basic MySQL Structure

- ***Database Server***
  - `hal` is the ***database server*** that we use in this class

- ***Databases***
  - In MySQL, synonymously called schemas

  - Contain the actual data that is accessed by users

  - A single server can have multiple databases

# What's in a database?

- In MySQL, a database contains four things:
  - ***tables***
  - ***views***
  - ***stored procedures***
  - ***functions***

- We are primarily concerned with tables, but we'll step through the others

# Database views

- A ***database view***, or view, is a table-like object accessible in the RDBMS

- Unlike a table, however, a view is not a static object

- Instead, a view is the output of a stored query, which is run when the view is accessed

# Why use a view?

- Presents data to users while keeping underlying storage normalized

- Takes up very little storage, as you are only storing the rules to create the view
  - Can have performance impact

- Can be used to have a set place to see aggregated results, without storing aggregations

# Stored Procedures

- A ***stored procedure*** is stored SQL code that can be run to accomplish specific tasks

- Stored procedures can be used to process data, and can have syntax beyond basic SQL

- Generally created by DBAs to accomplish some goal

# User Functions

- A ***function*** is an expression that is executed within a query
- SQL contains many functions by default (`avg`, `sum`, etc)
- ***User functions*** extend the defaults by letting users have their own
    - Won't be touching on these in class

# Tables

- The ***table*** is the object you are likely to interact with the most

- We covered tables, or ***relations***, in considerable depth in a previous unit
  - Won't rehash here

- However, important to know what MySQL groups as objects in a table

# Contents of a table

- ***Columns***: pretty straightforward
- ***Indexes***: stored as objects within the table, but separate from the columns
- ***Foreign keys***: constraints on column values
- ***Triggers***: code that is run in response to some event that occurs ("triggers it")

# What you need to know

- As ***consumers of data***, you will primarily work with tables, and perhaps with views
- Within a table, your focus will likely be on columns, rather than the other features
- If you move toward doing more data management / design, other items may come up

# Coding conventions

- Sometimes called a "style guide"

- Sets of rules about how to do things in a programming language
  - Naming variables
  - Structuring programs
  - Using syntax consistently
  - Comments and preambles

# Using conventions

- Often, your workplace or organization might have established conventions
  - This is a drag when they don't match yours
- Other times, you are working solo - need to establish your own
  - Prioritize clarity, consistency, and replicability
  - Look to what you do organically, then make it systemic

# What affects conventions?

- Obvious factors:
  - Clarity - your code should be easy to read and understand

  - Consistency - when you move from program to program, you should follow the same style

  - Replicability - when you program, you should do the same thing the same way within reason

# Other factors

- The IDE may affect your decision
    - Some IDEs may support "collapsing" code sections

    - Others have defaults for indentation, etc

- The language may have some requirements
    - Not every language has flexible rules about placing things

    - Sometimes, programs need certain formatting to work

# Laying out SQL code

- SQL is "friendly" toward adding whitespace to your code
  - This means you can use indentation and capitalization to make your code more clear

- Use the flexibility to make your code easier to read
  - Don't make things harder than they need to be!

# Some SQL conventions

- Write a preamble to your program
  - Not necessarily SQL-specific - just a good idea for programming in general
  - Describes the program name, author, purpose, dependencies, perhaps version (if not in a version control system)
  - Often encompassed by a comment box if the language supports that

# SQL conventions, contd

- Use capitalization consistently
    - In SQL, this means capitalizing SQL keywords, and leaving lowercase other items
    - Don't mix and match!
- Indent code and use carriage returns after each line
    - SQL allows you do other things, but this improves readability

# SQL conventions, contd

- Separate commands with carriage returns
  - Makes it easier to see where each command ends
- Use comments liberally to denote each block and to denote line-specific items
  - Comments are love, comments are life
- Place commands on multiple lines, indent to preserve vertical alignment

# Your conventions

- This class does not have an established style guide
  - When in doubt, you might copy some of how I write my code
- Ultimately, you have to choose your own style
  - If your code is confusing or disorganized, it could affect your grade

# Getting connected

- We'll use two methods to connect to MySQL in this class:
  - Command line
  - MySQL Workbench
- There are other ways to reach a server, which we may cover later
  - Other DB programs, programming language interfaces, ODBC

# Our MySQL Server

- We are working with a MySQL server located on `hal`
- This is the same place you were working in the Unix / Linux unit
- You each have a username to log in and manipulate the database

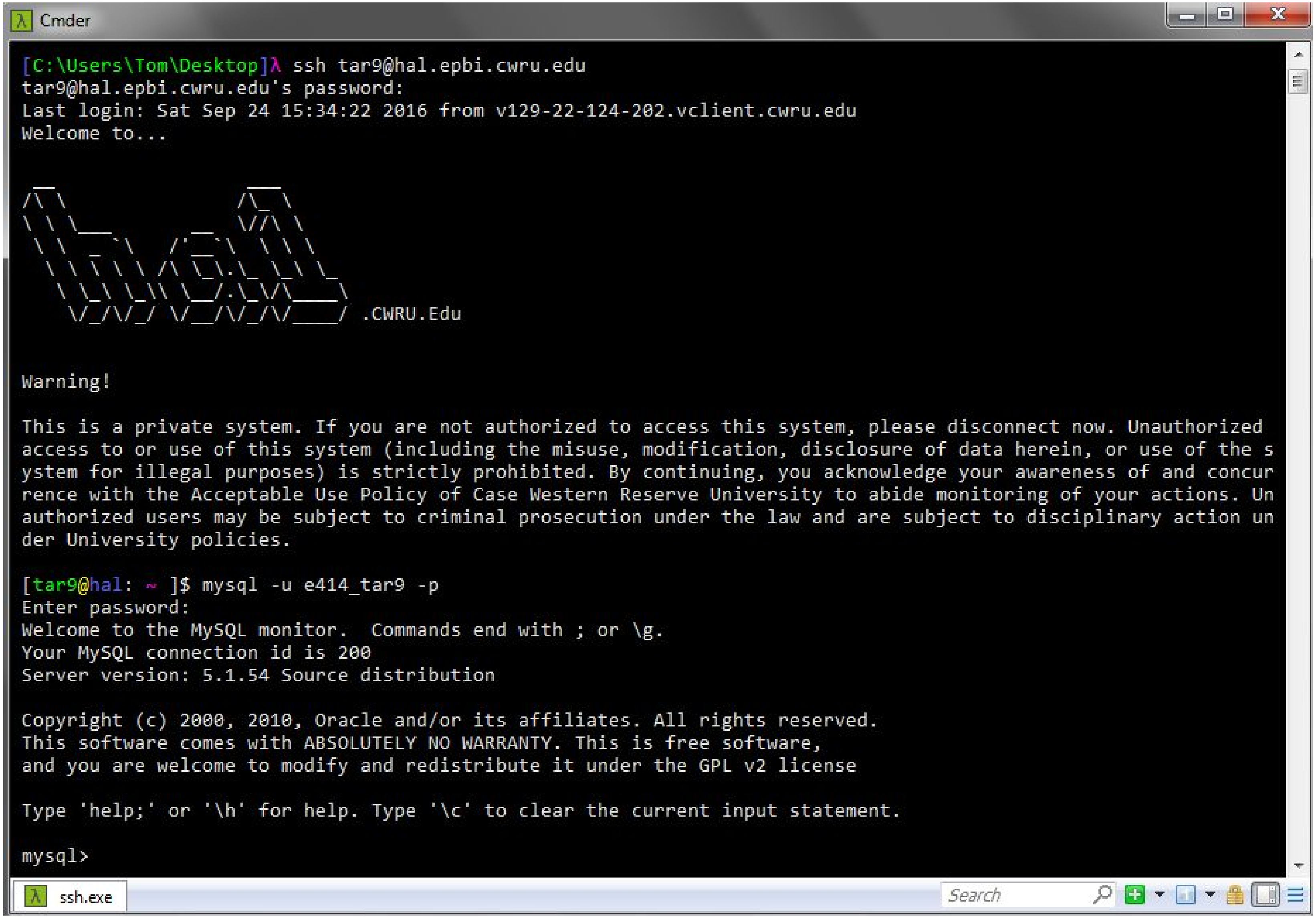
# Privileges

- For the time being, you are only able to read data from the SQL server
- This should prevent you from destroying anything by accident
- Later, we will expand your privileges so you can learn more about basic administration

# Using the command line

- The command-line interface can be accessed on `hal`
  - You are able to use the command-line interface from other machines - but for this class, we'll use `hal`

- The command line interface is very useful for running automated scripts, or processing data
  - Can also be used interactively



```
[C:\Users\Tom\Desktop]λ ssh tar9@hal.epbi.cwru.edu
tar9@hal.epbi.cwru.edu's password:
Last login: Sat Sep 24 15:34:22 2016 from v129-22-124-202.vclient.cwru.edu
Welcome to...


 __                 ___
/\ \               /\_ \
\ \ \___       __  \//\ \
 \ \  _ `\   /'__`\  \ \ \
  \ \ \ \ \ /\ \_\.\_ \_\ \_
   \ \_\ \_\\ \__/.\_\/\____\
    \/_/\/_/ \/__/\/_/\/____/ .CWRU.Edu


Warning!

This is a private system. If you are not authorized to access this system, please disconnect now. Unauthorized
access to or use of this system (including the misuse, modification, disclosure of data herein, or use of the s
ystem for illegal purposes) is strictly prohibited. By continuing, you acknowledge your awareness of and concur
rence with the Acceptable Use Policy of Case Western Reserve University to abide monitoring of your actions. Un
authorized users may be subject to criminal prosecution under the law and are subject to disciplinary action un
der University policies.

[tar9@hal: ~ ]$ mysql -u e414_tar9 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 200
Server version: 5.1.54 Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

# Getting connected on the command line

# Useful features

- Using the command line interface gives you flexibility

- You can use < to pass a file with SQL commands into the command line

- You can use | and > to redirect the output!
  - Just like the Unix / Linux unit

# Other notes

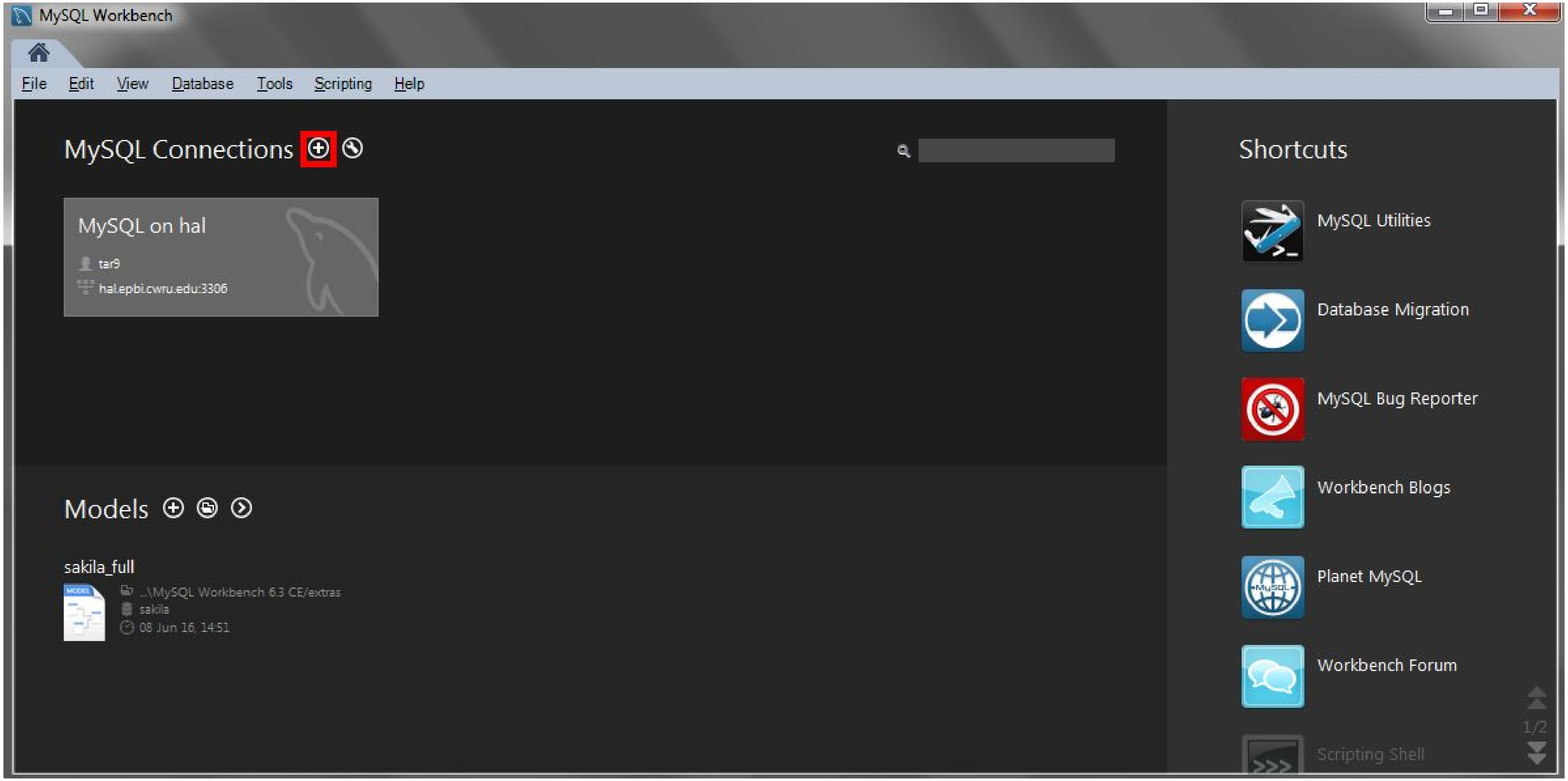
- Just like every Unix / Linux command, MySQL has a `man`ual
- There are a lot of options for using the MySQL command line tool
  - You might need to use these in the future
- MySQL command line + shell scripts: simple automated data processing

# Storing your password

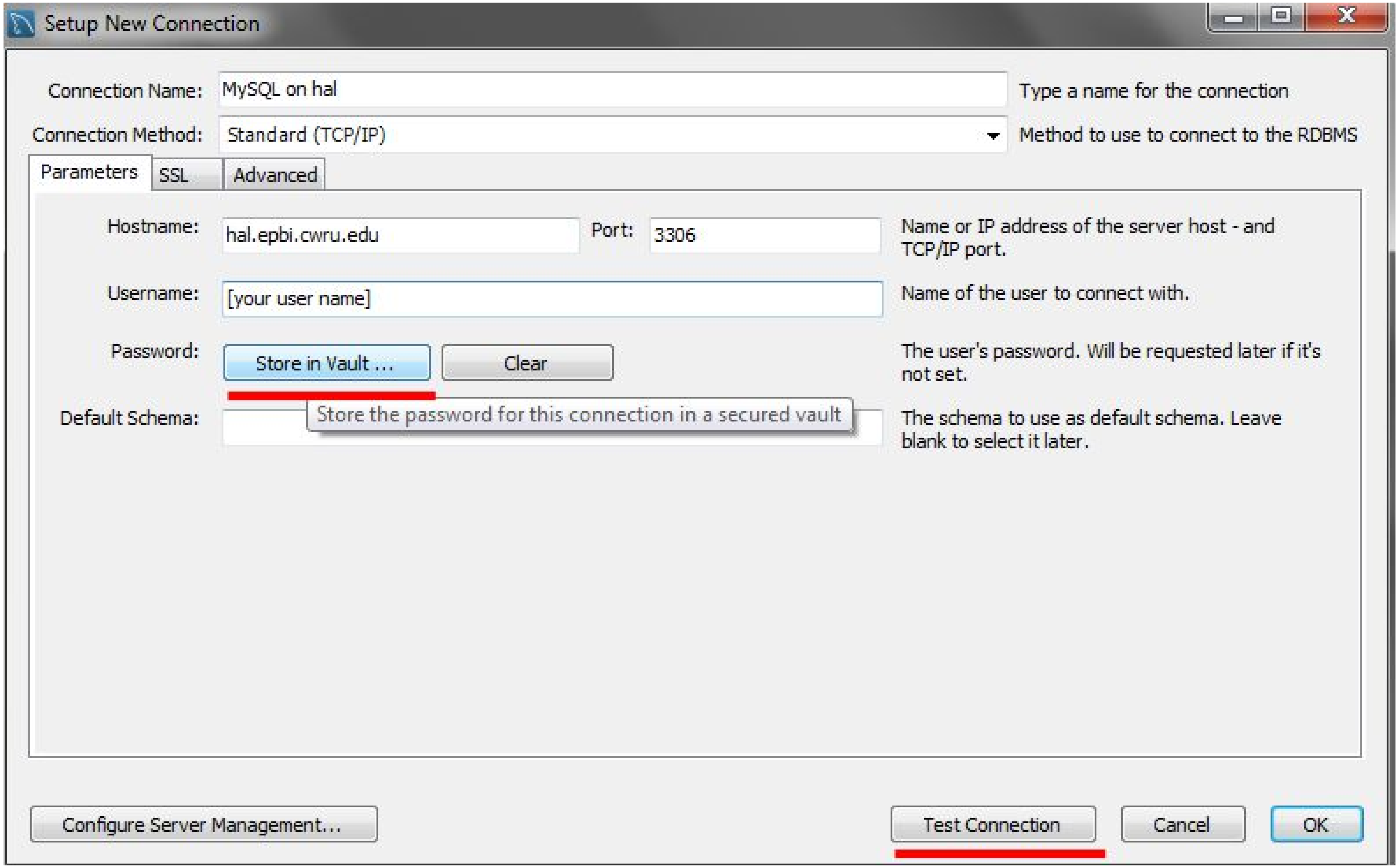
- It can be annoying to type the -u and -p each time
  - And to have to type in your password
- Can store this in a .my.cnf file in your home directory instead
  - Use the example version posted if you'd like
- Once done, you can just use `mysql`

# MySQL Workbench

- A product now owned by Oracle - you can download it from the MySQL website
  - Google "MySQL Workbench"
- Broadly, a tool to be used by DBAs / developers
- You can use it to perform queries and inspect database objects



# Add your connection



Setup New Connection

Connection Name: MySQL on hal — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Hostname: hal.epbi.cwru.edu Port: 3306 — Name or IP address of the server host - and TCP/IP port.

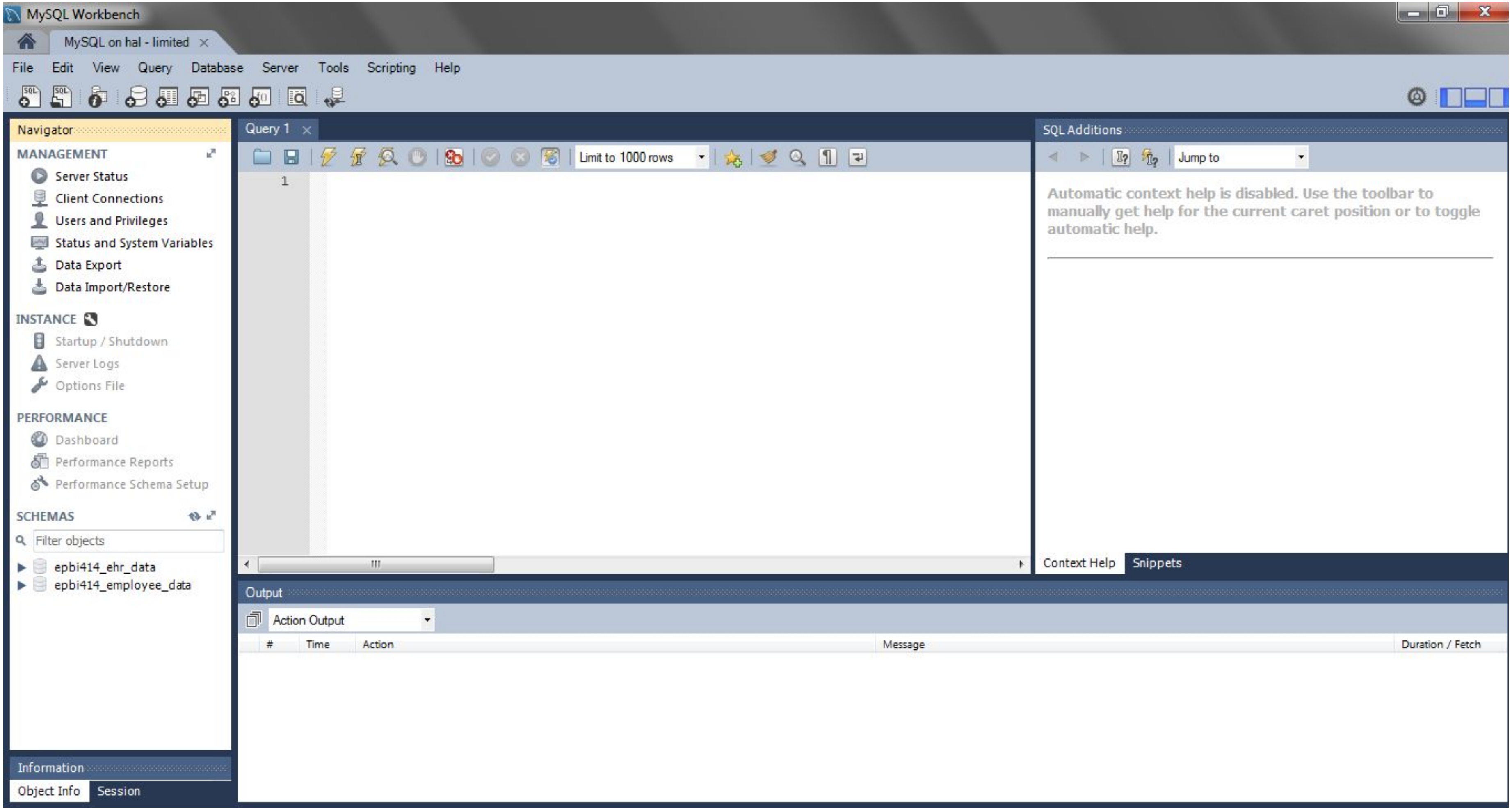
Username: [your user name] — Name of the user to connect with.

Password: — The user's password. Will be requested later if it's not set.

Store the password for this connection in a secured vault

Default Schema: — The schema to use as default schema. Leave blank to select it later.

## Store your password, and test your connection



Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

## Once connected, you see this

# Notes about Workbench

- You probably won't use everything it can do
  - Mainly for DBAs, architects, et cetera

- Allows you to select and execute code
  - Highlight code in the editor, then hit Ctrl-Enter (Windows) to run - might be Cmd-Enter on a Mac

- If you are typing and stop for a second, it will suggest things to tab-complete

# More notes on Workbench

- On the left side, you can navigate through all the databases to which you have access

- Lets you descend into multiple levels (tables, column, et cetera)

- Using this, you can also right-click and retrieve information about these objects

# Before we start coding...

- A suggestion to you: change whatever editor you use to put spaces in for tabs
- In MySQL Workbench, it's in Edit > Preferences
- In `joe`, you can use command option -spaces and -tab 4 (4 spaces for a tab)

# Why spaces?

- When you put a "tab character" into a document, you are inserting a ***hidden whitespace*** character

- You can't visually tell the difference between spaces and tabs - can make formatting go off

- Can cause problems crossing OSes

# SQL Programming

- SQL programming is done by executing SQL statements (sometimes SQL queries) against the database

- Queries can range from being very simple to being extraordinarily dense and complex

- Some basic SQL syntax that you should know...

# Basic SQL Syntax - 1

- ***Keywords:*** These are English words, reserved in SQL for operations
  - SQL does not care about case (capitalization)
  - Traditionally, you capitalize the keywords, and leave other things in lowercase (convention!)

    Example:

    ```
    SELECT column FROM table ...
    ```

# Basic SQL Syntax - 2

- ***Statements:*** SQL statements end with semicolons (;)
  - Sometimes, that's flexible, but you should subscribe to it anyway

- ***Whitespace:*** SQL ignores white spaces. The below are equivalent.

```
SELECT column FROM table ...
SELECT     column FROM      table ...
```

# Basic SQL Syntax - 3

- ***Comments:*** Comments are denoted using two dashes, then a space (-- ) and end at the end of the line

- A reminder: **Comment. Your. Code.**
  - It is good practice.
  - **<u>In this class, it will affect your grade.</u>**
  - It helps you understand what you were doing.
  - It helps others understand what you were doing.

# Basic SQL Syntax - 4

- ***Quotes***: ANSI SQL uses a single quote character (`'`) to denote strings

- You can sometimes get away with double quotes (`"`) depending on your RDBMS...

- But best not to get into the habit

# Break Time

# Your first SQL command

- One of the key tasks in SQL is retrieving data

- This is done using a `SELECT` statement

- There are many additional options that you can use in a `SELECT` statement
  - Filtering, aggregating, et cetera

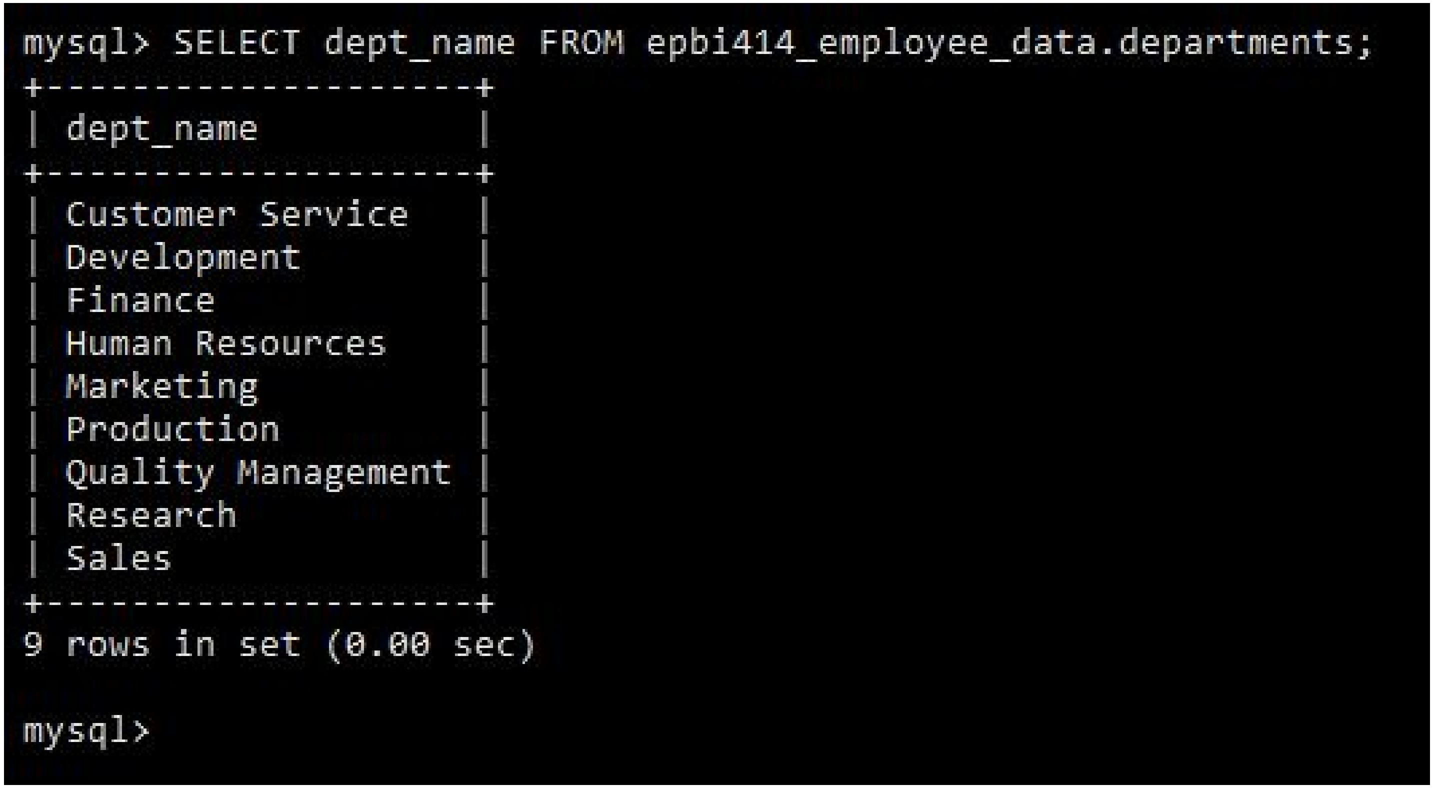
# Basics of SELECTing

- The most basic syntax of `SELECT` is:

```
SELECT column1,column2
       FROM database.tablename;
```

- This would return the values of columns 1 and 2 from `tablename` in `database`.

# An example SELECT



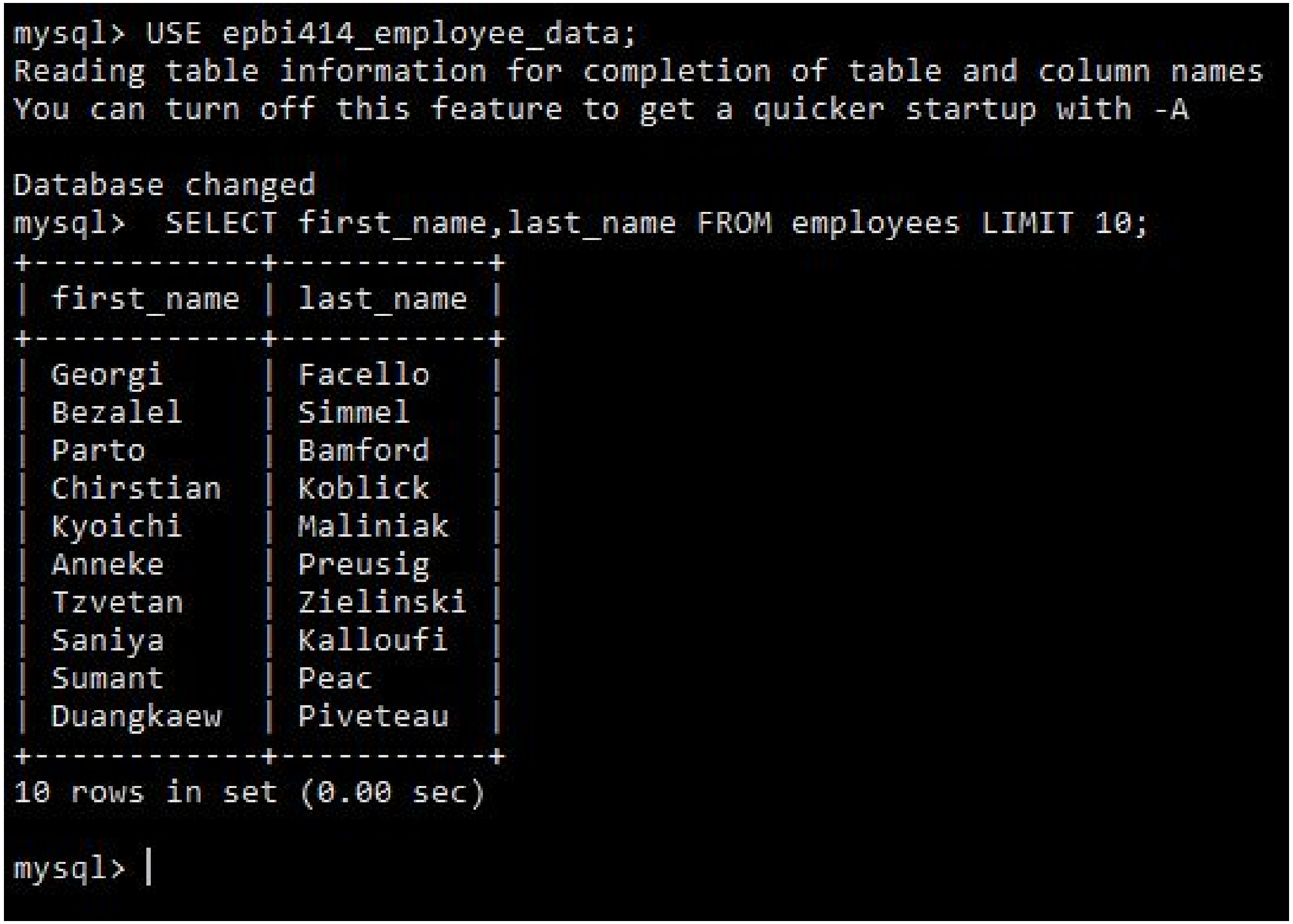
```
mysql> SELECT dept_name FROM epbi414_employee_data.departments;
+--------------------+
| dept_name          |
+--------------------+
| Customer Service   |
| Development        |
| Finance            |
| Human Resources    |
| Marketing          |
| Production         |
| Quality Management |
| Research           |
| Sales              |
+--------------------+
9 rows in set (0.00 sec)

mysql>
```

# USE and LIMIT

- Before we go further, we should talk about two useful commands
  - Somewhat MySQL-specific
- `LIMIT` controls how many rows are returned from a query
  - Useful when you are working interactively
- `USE` controls which database is queried by default

# LIMIT and USE



```
mysql> USE epbi414_employee_data;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>  SELECT first_name,last_name FROM employees LIMIT 10;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Georgi     | Facello   |
| Bezalel    | Simmel    |
| Parto      | Bamford   |
| Chirstian  | Koblick   |
| Kyoichi    | Maliniak  |
| Anneke     | Preusig   |
| Tzvetan    | Zielinski |
| Saniya     | Kalloufi  |
| Sumant     | Peac      |
| Duangkaew  | Piveteau  |
+------------+-----------+
10 rows in set (0.00 sec)

mysql> |
```

# SELECTing it all

- If you want, you can return every column from a table by using the *

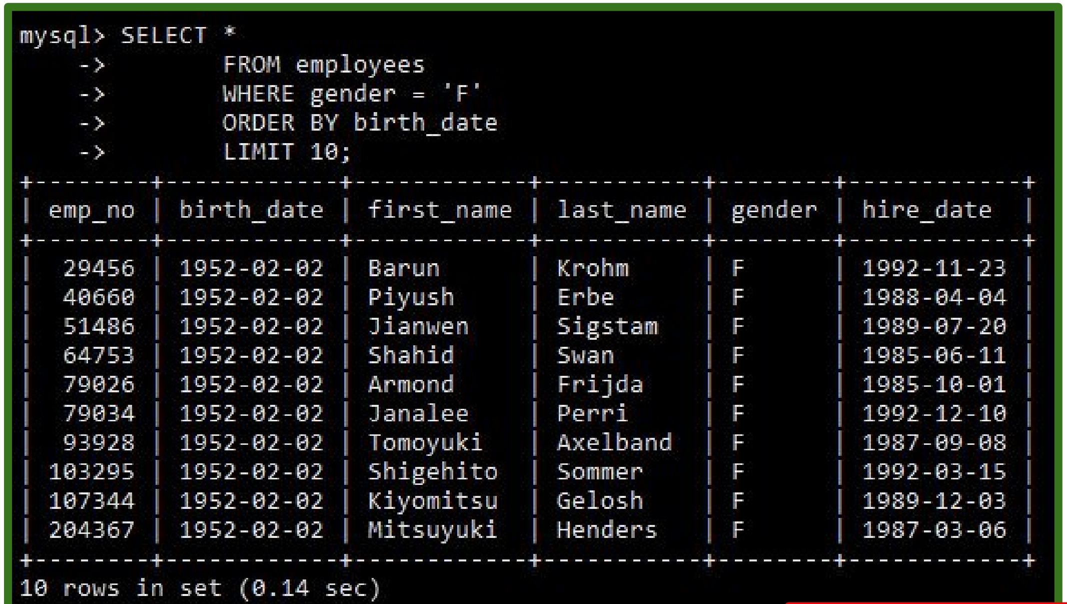
- Be warned that this will impact performance
  - Maybe don't do this unless you need

- But for initial exploration, getting all the columns helps you understand a table

# Sorting and filtering

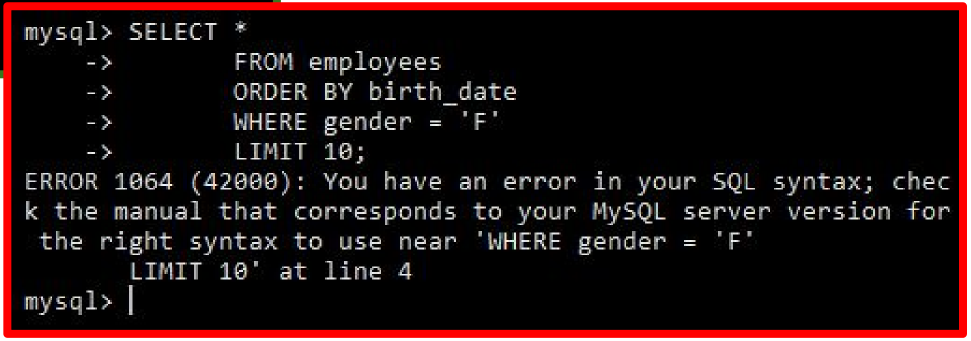
- You can sort your data by using the `ORDER BY` statement
  - Use `DESC` to get descending ordering
- You can filter your data using the `WHERE` statement
  - Many ways to filter - coming up!
- You have to filter before you sort!

# Command order matters!



```
mysql> SELECT *
    ->        FROM employees
    ->        WHERE gender = 'F'
    ->        ORDER BY birth_date
    ->        LIMIT 10;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  29456 | 1952-02-02 | Barun      | Krohm     | F      | 1992-11-23 |
|  40660 | 1952-02-02 | Piyush     | Erbe      | F      | 1988-04-04 |
|  51486 | 1952-02-02 | Jianwen    | Sigstam   | F      | 1989-07-20 |
|  64753 | 1952-02-02 | Shahid     | Swan      | F      | 1985-06-11 |
|  79026 | 1952-02-02 | Armond     | Frijda    | F      | 1985-10-01 |
|  79034 | 1952-02-02 | Janalee    | Perri     | F      | 1992-12-10 |
|  93928 | 1952-02-02 | Tomoyuki   | Axelband  | F      | 1987-09-08 |
| 103295 | 1952-02-02 | Shigehito  | Sommer    | F      | 1992-03-15 |
| 107344 | 1952-02-02 | Kiyomitsu  | Gelosh    | F      | 1989-12-03 |
| 204367 | 1952-02-02 | Mitsuyuki  | Henders   | F      | 1987-03-06 |
+--------+------------+------------+-----------+--------+------------+
10 rows in set (0.14 sec)

mysql>
```



```
mysql> SELECT *
    ->        FROM employees
    ->        ORDER BY birth_date
    ->        WHERE gender = 'F'
    ->        LIMIT 10;
ERROR 1064 (42000): You have an error in your SQL syntax; chec
k the manual that corresponds to your MySQL server version for
 the right syntax to use near 'WHERE gender = 'F'
      LIMIT 10' at line 4
mysql> |
```

# Filtering with WHERE

- The `WHERE` statement is used to filter the results of a `SELECT` statement

- Filtering using `WHERE` means understanding basic programming / symbolic logic

- We will cover those in greater depth in SAS as well

# Control statements

- In programming, a ***control statement*** is a statement which determines whether other statements are executed

- These generally break into two groups: ***if statements*** and ***loops***

- We will talk about if statements here
  - Loops come later

# If statements

- Generally speaking, a programming ***if statement*** has this format:

  If **[some test]** is true, then do **[this thing]**.

- Can include an "otherwise" too...

  If **[some test]** is true, then do **[this thing]**.
  Otherwise, do **[this thing]**.

# If - then - else

- These statements are usually referred to as **if**, **then**, and **else**.

- Can be combined using ***else if*** - allows chaining multiple conditions
  - If x = 1, then do j

    Else if x = 2, then do k

    Else do m.

- Why does this matter here?

# Truth and falsity

- The WHERE statement is used to return a row ***if*** some conditions are met
- So, we need to understand how to write basic if statements
- This means knowing a few basic logical operators

# AND, OR, NOT

- SQL makes use of three basic logical operators: `AND`, `OR`, and `NOT`

- Once we get those down, we'll discuss `IN`, `LIKE`, and `BETWEEN`, as well as `NULL`

- These, along with basic mathematical tests, give us a ton of filtering options in SQL

# Logical AND

- A logical AND evaluates to TRUE when all of the conditions are TRUE.

  IF John is Male AND John is from Long Island, THEN print "Hello, John from Long Island!"

- Only works when John is both male, and from Long Island.

# Logical OR

- A logical OR evaluates to TRUE when at least one of the conditions are TRUE.

  IF Jane is a master's student OR Jane is a doctoral student THEN print "Jane, you are a graduate student."

- An OR is only FALSE when all the conditions are FALSE.

# Logical NOT

- A NOT statement, sometimes called a negation, turns a TRUE into a FALSE and vice versa

  IF Jane is NOT a master's student, THEN send Jane a promotional letter.

- Sometimes, you might put the NOT outside the regular statement (IF NOT Jane...)

# Logical tests

- SQL gives us a few different logical tests:
    - Equality (`=`)
    - Non-equality (`<>` and sometimes `!=`)
    - Greater than and less than (`>` and `<`)
    - Greater than/less than or equal to (`>=` and `<=`)
    - Within an inclusive range (`BETWEEN`)
    - Within a discrete set (`IN`)
    - Matching a pattern (`LIKE`) - uses % and _ as wildcard

# Example filter logic

```
SELECT  *
        FROM employees
        WHERE (birth_date BETWEEN '1950-01-01'
                                 AND '1953-12-31'
            AND (emp_no < 10050 OR emp_no > 11000))
            OR first_name LIKE 'Chir%';
```

- Picks people who either have birthdays between 1950 and 1953, and employee numbers less than 10050 or greater than 11000, OR...

- First names that contain the string Chir plus any number of characters

# Wildcard matching

- The `LIKE` statement lets us search text using wildcards
    - Similar to what you learned about regular expressions

- The two wildcard symbols we will discuss are `%` and `_`

- Wildcards are expensive - don't use them if you don't need them

# Using % to match

- The `%` symbol matches ***any number of occurrences*** of ***any character in its location*** in the search pattern

- For example, "**%ello**" matches **hello**, **Hello**, **Othello**, but not **Mellon**.
  - Also matches zero: ello would be included.

# Wildcard % match example

```
SELECT  first_name,
        last_name
        FROM employees
        WHERE first_name LIKE 'Geoff%y'
        LIMIT 3;
```

- This returns
  - 'Geoffry','Noriega'
  - 'Geoffry','Ranka'
  - 'Geoffrey','Claffy'

# Using _ to match

- The underscore (_) matches ***exactly one*** character
- Will not match 0
- Will only match the precise number of spaces you have

# Wildcard _ match example

```
SELECT  first_name,
        last_name
        FROM employees
        WHERE last_name like "___sen"
        LIMIT 3;
```

- This returns
    - 'Jianhao','Thisen'
    - 'Inderjeet','Jansen'
    - 'Jixiang','Thisen'

- Using “%sen” would also match “Rosen” and many more

# Some notes

- SQL always evaluates the AND statement before it checks the OR statements
  - To override this behavior, use parentheses

- `BETWEEN`, `IN`, and `LIKE` might not work consistently across data types
  - Being between A and Z may not work

- The `AND` in a `BETWEEN` statement isn't the same as a logical `AND`

# NULL

- In SQL, a missing data value is usually set to NULL

- Recall that this is a case where system missing vs. actual missing can be a problem
  - Is NULL there because we forgot to enter the data?

- NULL does not work quite like the other operators

# Checking NULLness

- You cannot check if something is NULL using an equality operator (`x = NULL`)

- Instead, you use `IS NULL` and `IS NOT NULL`

- Otherwise, the logical operations remain the same (can be combined using `AND`, `OR`, et cetera)

# A few other useful notes

- Especially at the command line, you will find the `SHOW DATABASES` and `SHOW TABLES` commands useful

- Lets you find out what you have access to, even without having a graphical interface like MySQL Workbench

# JOINs

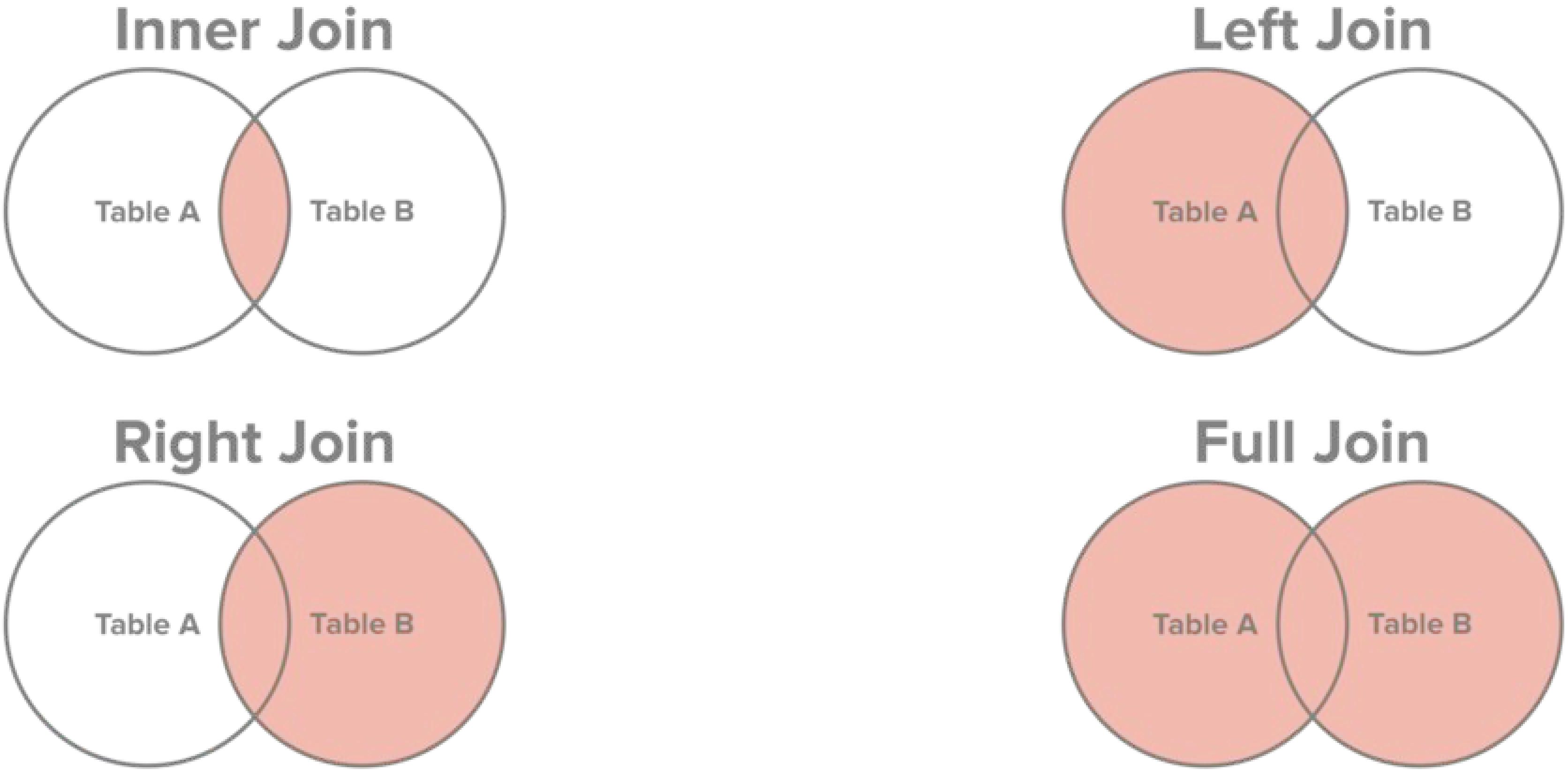
- By nature, the relational model stores data in multiple tables
  - Kind of the whole point
- Humans like to see the data in one place
- How do we bring that data together? We use ***joins***

# Types of JOIN

- SQL has four basic types of join:
    - Inner join
    - Left join (sometimes left outer join)
    - Right join (sometimes right outer join)
    - Full join (sometimes full outer join)

- The easiest way to understand these is by way of Venn diagrams

# JOIN Venn Diagrams



Taken from http://www.sql-join.com, license unclear.

# Inner joins

- For today, we'll focus on the humble inner join
  - Generally, the join you get if you don't otherwise specify
- The inner join is sometimes called an equijoin
  - Returns rows only when they are found in both tables, hence, equijoin

# Subqueries and naming

- Generally, you want to join together the results of two or more different queries
    - So you can filter things in each table, et cetera

- We can accomplish this using a ***subquery***, which is denoted by a ()

- Subqueries (and indeed tables) can be assigned ***aliases*** to make things more clear

# An example join

```
SELECT  e.emp_no,
        e.first_name,
        e.last_name,
        e.hire_date,
        e.gender,
        s.salary
        FROM employees AS e
        INNER JOIN
            (
            SELECT  emp_no,
                    salary
                    FROM salaries
                    WHERE to_date = '9999-01-01'
            ) AS s
        ON e.emp_no = s.emp_no
        LIMIT 10;
```

# WHERE vs. ON

- When you join, you give the SQL engine a variable to join on
    - Technically, you give it a logical condition on which to join - they can have multiple steps
- It is important to note that `WHERE` is not the same as `ON`
- For an inner join, they are synonymous
    - Not always - more to come!

# Acknowledgements

1. My thanks to Omar De la Cruz Cabrera for the initial version of these slides, used in Fall 2015

2. SQL Venn Diagrams taken from www.sql-join.com