Practical SQL

Session 01

Tuesday 01 October 2013

General Assembly

Welcome

- Class focus is SQL syntax
- We will also cover:
  - Enough background to orient thinking
  - Usage of SQL to explore and analyze data
- Exercises away from class build familiarity

Who are you and why are you here?

- What sort of companies do you work for?
- What are your roles?
- What are you using for analysis?
- What are you interested in?
  - Replicating Excel analyses with SQL
  - Running queries
  - Orientation to SQL and database concepts

Who Am I

- Entrepreneur / consultant
  - – Data management
  - – CRM migration and integration
- Self-taught on LAMP(ython) stack
    - Plus some DNS, bash scripting, XML / XSLT
- Prior professional experience:
  - – Office of Management and Budget
  - – Investment banking (JP Morgan, UBS, boutique)
  - – MBA, University of Chicago

Roadmap – Session 1

- Preliminaries
- What is SQL
  - What it is used for
  - What is meant by it
- Software and sample database installation
- Loading data from CSV
- Query syntax basics
  - Running queries
  - Filtering, grouping
  - Functions
  - Combining tables
- Using queries for analysis

Tools

- Database Software
  - Some means of running SQL queries is needed
  - MySQL, Postgres, Sql Server, Oracle
    - Microsoft Access is NOT recommended
    - If you have questions about your SQL implementation, see me
  - MySQL is free, download from mysql.com
    - MySQL
      - Both “Community Server” and startup package
    - MySQL Workbench
- Text editor – you may or may not need one
  - Wordpad, Text Edit are very basic examples
  - Textmate, Sublime Text are more featured tools
    - Windows: Notepad++
  - Do NOT use a word processor (e.g. Word)
  - Do NOT use ‘rich text format’ / *.rtf
  - SQL code should just be ASCII text, with NO formatting information

Get A Database Program

- Some means of running SQL queries is needed
    - MySQL, Postgres, Sql Server, Oracle
        - Microsoft Access is NOT recommended
        - If you have questions about your SQL implementation, see me
    - MySQL is free, download from mysql.com
        - MySQL
            - Both "Community Server" and startup package
        - MySQL Workbench

What is SQL -- Usage

Role of SQL in Data Analysis

- Data analysis requires:
  - Getting, cleaning, loading, ‘modeling’
  - Exploring, manipulating, extracting, analyzing
  - Reporting, visualization
- SQL is best at analysis, but call help with all of this

Getting and Cleaning

- Something else has to get the data to CSV or database
- SQL string functions can help clean and reformat data
- Tricks with LOAD and INSERT can remodel data

Reporting and Visualization

- SQL has zero visualization capacity, but it can output data as needed for your graphics program
- “Reporting” is here sort of a term of art
  - E.g. pivot tables, cross tabs, and so forth
  - SQL can do an awful lot, but isn’t great at this
  - Implementations do provides base layers to reporting tools like Jasper, Pentaho, etc

Analysis and Transformation

- SQL's stock in trade is filtering, aggregating and grouping on set-based criteria
  - Filter WHERE records meet some stated criteria
    - State equals Illinois, name in (Smith, Baker, Han)
    - Date before 2012-06-01
    - Score greater than 15
  - GROUP BY values in a field, or in a combination of fields
- Uses that capacity to combine information in different sets / tables by matching records on some criteria
- We can nest these abilities into "subqueries", generating criteria on the fly

As A Language – Application Usage

- I.e., why we care
- A syntax for:
  - Describing tables
  - Extracting and manipulating data from those tables
  - Plus dealing with the cruft of managing data: loading, indexing, etc
- The action is in bullet #2, but we have to deal with the other stuff too

Extracting and Manipulating

- Extracting:
  - We want to specify criteria for which records we display or analyze
  - Combine data extracted from several tables
- Manipulating:
  - Transform values in fields
  - Calculate values for fields of groups of records
- These fundamental operations can be extended to very complex transformations

What is SQL -- Architecture

A “Standard”, Not a Program

- There are many SQLs
  - MySQL / Postgres / SQLite / Microsoft Access / Oracle / Sybase / etc etc etc
- Each is an implementation of a standard
  - “SQL” is basically a program’s promise to behave in a defined way in response to a particular query for a particular set of data

“Client / Server”

- Most implementations are actually two programs
  - One is a front end / client, giving us access to a server and showing its outputs
    - MySQL Workbench
  - The other is a back end / server, holding the data and processing the queries
    - MySQL
  - MySQL Workbench is to MySQL as Chrome browser is to an HTTP server

Client / Server Implications

- We can access existing data
  - We don't need data dumps to do analysis
    - We do need some sensible and safe access to the existing databases
  - If your production database has a million records, you can analyze them all without downloading them all
- We can access remote data
  - If your servers are on Rackspace or Amazon you can still get at the data
- Programs can access the data
  - Languages like Ruby, Python, PHP all have tools for interacting with databases
  - They act as clients, but getting data for (say) serving web pages rather than display to an analyst

Excel vs SQL

Why Excel?

- Good graphic interface orients us to the data and the tools
  - We can find tools just by clicking around the menus
- Excel organizes models visually, so we can _see_ the relationships among cells
  - We can see the cells flowing into a formula
  - We imagine extensions of our models by thinking through new visual relationships
- Excel handles many issues we don’t even know about
  - Until we use a tool with less hand-holding – like SQL

Why SQL? -- Analytic

- Operated through text statements
    - Easy to identify and extend patterns
        - Replicable
            - If a new query resembles a prior one, just copy and revise the prior
        - Chainable
            - We can easily add conditions and "filters"
            - We can write a query and refer to its results in another query
        - Traceable
            - We can see how we got to a result by examining the query
- Non-visual modeling allows us to fluidly work in more than two dimensions

Implications

- Excel gets used like a Swiss Army Knife
  - We have problems, Excel gives us path to solve them
- SQL can do the same things and more, BUT requires infrastructure, training and problem solving – i.e., brain damage
  - When we do need it, who has time for training?

Why SQL? -- Organizational

- A lot of data already in your company's relational databases
- Dev team familiar with SQL
- SQL easier to automate
- Easy to operate over a network

Getting Started

Getting Files

- Download files from https://github.com/chernevik/practical_sql
  - Click ‘Download ZIP’ to get an archive of the files
  - Open / unpack archive

Install Database Software

- If you don't have access to a database, follow instructions in files
- Note that you must install both a client program and a server program
  - Probably, also, a script for starting the server on computer boot
- Reach out to me if you have trouble

Install Sample Database

- In files, find script ‘setup_sampdb.sql’
- Just run this in your MySQL client
  - This is MySQL specific; tell me what you are using and I’ll provide scripts to load from CSV
- If you read this file, it’s just a big query that creates a database and tables, and inserts data

Queries

Run a Query

- In your client, type:
  - ‘SELECT 2 + 2;’ and ‘execute’
- Note that queries are synonymous with ‘commands’
  - We use them to get information about the databases and tables available
  - We can test out some functions and expressions with them

Queries to Explore Syntax

- SQL evaluates more sophisticated expressions, like logic:
  - SELECT 1 AND 2 -> 1
  - SELECT 2 AND 2 -> 1
  - SELECT 0 AND 2 -> 0
- It supports functions:
  - SELECT LEFT(‘cat in the hat’, 3) -> ‘cat’
  - SELECT RIGHT(‘cat in the hat’, 3) -> ‘hat’
  - SELECT UPPER(‘cat’) -> ‘CAT’
  - SELECT LENGTH(‘cat in the hat’) -> 14
  - SELECT IF(1, ‘OK’, ‘NOT’) -> ‘OK’
  - SELECT IF(0, ‘OK’, ‘NOT’) -> ‘NOT’
- Each of these character strings is a SQL ‘statement’, made up of ‘expressions’
- The database program evaluates each expression, and ‘returns’ a result for it
  - Those results may become input to another, enclosing expression
  - SELECT IF(0 and 2, ‘OK’, ‘NOT’) -> ‘NOT’
- We rarely use SQL this way, but it does show how expressions and functions work

How do we retrieve data?

- So:
  - 'SELECT * FROM sampdb.student'
  - This query / statement gets all fields from the table student in the database sampdb
  - We will add more clauses to this to control its output

Queries are text, period

- Whether entered by typing, or loaded from a file, or loaded in some other way, the client only cares that it has text to pass to the server
- For example, in the downloaded materials, find:
  - sessions/01/simple.sql
  - Open in query editor
  - Execute

Aside: Organizing Your Environment

- Queries are typically written in text files
  - Again, no word processing, no .doc or .rtf files
- Make a place on your computer to hold your files – data, queries, drafts
  - Your files can expand quickly, as you copy files to experiment or to make variations
  - Use of subdirectories helps keep things organized
  - Code repository tools like Git and Mercurial make a lot of sense
    - This would be a great time to begin learning and using these

Loading Data

Annoying but necessary

- No fundamental concepts involved here – just file management
- Understanding this process does orient us to the fundamentals of what the computer is doing
- We will go over this fast and revisit in exercises
  - Next week we'll go over problems in those

First, get data to database

- Get data into CSV format
  - Excel can do this for us
  - On Mac:
    - “Comma Separated Values” will have CR line terminators
    - “Windows CSV” will have CRLF terminators
- Line terminators? Hang on

Create a database

- Simple query creating a home for our table

```
DROP DATABASE IF EXISTS learning;
CREATE DATABASE learning;
```

Create a table

- Very simple table
- Notice homogeneity of data type (all varchar)

```
DROP TABLE IF EXISTS learning.from_excel;
CREATE TABLE learning.from_excel (

   date varchar(20),
   category varchar(20),
   event_id varchar(20)
);
```

Load data

- Let’s break this down
  - CR = ‘carriage return’
  - LF = ‘line feed’

```
LOAD DATA LOCAL
   INFILE "/Users/gimli/Desktop/from_excel_windows.csv"
   INTO TABLE learning.from_excel

   FIELDS
      TERMINATED BY ','
      ENCLOSED BY '"’

   LINES
     TERMINATED BY '\r\n'   -- for files with CRLF terminators
     -- TERMINATED BY '\r'       --  for CR terminators

IGNORE 1 LINES;
```

A moment on file data structure

- Loading CSV to SQL forces us to consider how our data is represented in a file
- Excel handles a variety of file storage formats for us – it will guess which is the proper one
- SQL will not
  - It is a power user tool
  - Stuff like file format has defaults but if these don’t work we must figure things out for ourselves

File data structure: Problem

- The file is a string of bytes – in our simple case, ASCII data
  - Unicode? Wrong class
- Some of those bytes are data, some are metadata
  - How does MySQL know where one field begins and another ends?
  - Where one record ends another begins?

Solved by convention and definition

- Fields “delimited”, or “terminated”, by commas
  - When MySQL sees a comma, it starts a new field
- Fields are also “enclosed” by double quotes
  - When MySQL sees a double quote, it considers all that follows a field until it sees another double quote
- Not redundant
  - This structure allows us to have commas in our data
  - If an enclosure is missing MySQL will just go with the field terminator
  - Double quotes in data can be handled by escaping mechanism (e.g. \” is not an enclosure) but we haven’t time for that

Solved by convention and definition - 2

- Lines (e.g. records, rows) terminated by a line termination character(s)
  - In Windows, generally carriage return + line feed, or CR LF (‘\r\n’)
  - In Mac and Linux, line feed (‘\n)
- These characters are VERY ANNOYING because we almost never see them
  - Our programs all have facilities for interpreting them for us
- We can figure this out or just guess

NEVER MIND -- JUST DO THIS

- Create a database
- Create a table
- Save as CSV
- Run this
- Check results, if bad fiddle with this and try again

```
LOAD DATA LOCAL
   INFILE "/Users/gimli/Desktop/from_excel_windows.csv"
   INTO TABLE learning.from_excel

   FIELDS
      TERMINATED BY ','
      ENCLOSED BY '"’

   LINES
     TERMINATED BY '\r\n'   -- for files with CRLF terminators
     -- TERMINATED BY '\r'      --  for CR terminators

IGNORE 1 LINES;
```

Query Syntax

Simple Query

```
SELECT name, sex, student_id
FROM sampdb.student;
```

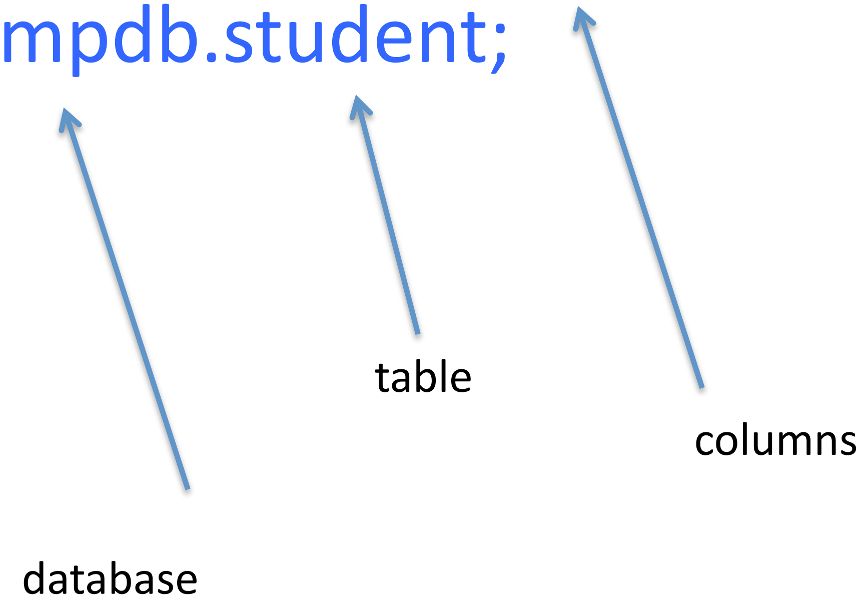
- Gets all fields of all records in table

Let’s unpack that:

- “database”
  - A program concept for holding “tables”
  - Comparable to a folder in your computer’s file system
- “table”
  - A program concept holding data about instances of a particular entity type – students, presidents, exams
  - These data are held in “records” or “rows”
- “record”
  - Data about a particular instance – about a particular student or particular movie
  - Data is in a collection of fields or columns
  - Comparable to a row in a spreadsheet
- “Column” / “field”
  - An attribute of an entity type about which data is collected – a student’s name, or sex, or the state of origin of a president
  - Comparable to a cell in a spreadsheet row

Simple Query

```
SELECT name, sex, student_id
FROM sampdb.student;
```



table

columns

database

We could also use:
SELECT * FROM sampdb.student
- << * >> is a wildcard meaning “all columns”

Filtering the results

```
SELECT
    *
FROM
   sampdb.student
WHERE
   sex = “F”;
```

- Notice the reorganization of the text for clarity
- WHERE clause defines conditions imposed on returned rows

An Aside on “query files”

- MySQL clients are looking for text only – e.g. files containing only ASCII or Unicode
- Don’t use Word, don’t use “rich text format”
  - These introduce characters that will confuse the MySQL client
- Use TextEdit, SublimeText, MacVim
  - vi !!!
  - A “text editor” of some sort

Filtering the results by ‘tuples’

```
SELECT
    first_name,
    last_name,
    state

FROM
    sampdb.president

WHERE
    (last_name, state) IN

    (
        ('Adams', 'MA')
    )
;
```

- A ‘tuple’ is a set of fields
  - A table is actually a set of tuples
- The IN operator allows us to see if a particular group of fields has a particular set of values

Analyzing the results

```
SELECT
    COUNT(student_id)
FROM
   sampdb.student
;
```

- COUNT() is a “function”
- Functions return information about or manipulate contents of a field
- Some functions apply only to field of single record
- Others, like COUNT, apply to the field for multiple records
- COUNT returns the number of not NULL values in the field
  - We’re ignoring “NULL” for now

Analyzing the results -- Aggregation

```
SELECT
    sex,
    COUNT(student_id),
    COUNT(sex)
FROM
   sampdb.student
GROUP BY
   sex
       ;
```

- COUNT() is an a “aggregate” function
- We can segment the records on which it works
- GROUP BY clause gathers up records by their value in a given field
- The function can be applied to any field, not just that by which records are grouped

Analyzing the results -- Prettify

```
SELECT
    sex AS Gender,
    COUNT(student_id) AS “ID Count”,
    COUNT(sex) AS gender_count
FROM
   sampdb.student
GROUP BY
   gender
   -- sex

  -- this is a comment
  -- all on this line after “--” is ignored

    # so is this, for MySQL
    /*
        And this, for MySQL
   */
;
```

- Inside the column expressions we’ve added “as [name]” clauses
- These are “alias” terms
- The results aren’t changed but their display is
- We can use the alias to specify the column

Functions

```
SELECT
   first_name,
   last_name,
   state,
   ROUND(DATEDIFF(death, birth) / 365, 1)
 AS age

FROM
   sampdb.president

ORDER BY
   age DESC

;
```

- DATEDIFF() is a defined MySQL function returning the number of days between two dates
- ROUND() controls decimal points
  - (we need this here to make some later code work)

More Functions, Aggregated

```
SELECT
  state,
  MAX(
    ROUND(DATEDIFF(death, birth) / 365, 1)
  )
 AS max_age,
   COUNT(*)

FROM
   sampdb.president

GROUP BY
   state

ORDER BY
   max_age DESC


;
```

- MAX() is an aggregate function, returing the largest of a set of values
- Here the set is the values calculated for a group of rows
- GROUP BY defines the rows
- We drop the name fields because they aren’t meaningful when rows are GROUPed

Find Me

- gordon@practicalhorseshoeing.com
- Twitter: @gordonagress
- github.com/chernevik/practical_sql
- 917-620-3402