SQL Basics

So far, we have defined database schemas and queries mathematically. We are very well prepared to learn SQL a programming language that allows us to execute our queries in a DBMS.

SQL is short for "Structured Query Language", but it is for more than just writing queries. It has two sub-parts:

- DDL (or "Data Definition Language"), used for defining schemas
- DML (or "Data Manipulation Language"), used for writing queries as well as for modifying the datase.

PostgreSQL

We'll be using an open-source DBMS called PostgreSQL. You'll find documentation for postgreSQL at the **postgreSQL home page (https://www.postgresql.org/)** including information on how to download it for your own platform, should you choose to.

These specific pages are highly relevant: - [PostgreSQL 9.1 Documentation] (http://www.postgresql.org/docs/9.1/static/index.html), particularly [Part II: The SQL Language] (https://www.postgresql.org/docs/9.1/static/sql.html), which explains the syntax and semantics of SQL. This documentation is concise and easy to read, so you'll want to reference it often. - **PostgreSQL Tutorial (http://www.postgresqltutorial.com/)**

What about other DBMSs?

Learning PostgreSQL will prepare you well for using other DMBSs, such as MySQL and Oracle. Unfortunately, there are differences in the SQL language across DBMSs. There is an official SQL standard that fully specifies the language, and is updated periodically. However, DBMSs typically don't implement every feature in the standard, and often add their own features as well. This means that you may find differences around the edges of the language when you work with a new DBMS. It also creates issues for portability of code.

Basic SELECT-FROM-WHERE queries

Before we begin, here is the university schema we have been working with (the key of each relation is in bold):

- Student(**sID**, surName, firstName, campus, email, cgpa)
- Course(**dept, cnum**, name, breadth)

- Offering(**oID**, dept, cNum, term, instructor)
- Took(**sID, oID**, grade)
- *and the foreign key relationships that you will recall*

We'll learn later how to express this in SQL using the Data Definition Language. But first, let's start writing some queries!

The simplest kind of query is a select-from-where. Here is an example:

```
csc343h-prof=> SELECT name             -- choose the column called "name"
csc343h-prof-> FROM Course             -- from the Course table
csc343h-prof-> WHERE dept = 'CSC';     -- choose only rows that satisfy
           name
---------------------------
 Intro to Databases
 Software Design
 Intro to Comp Sci
 Data Struct & Anal
 Intro to Visual Computing
 COBOL programming
(6 rows)
```

`WHERE` is equivalent to σ in relational algebra. Sadly, the designers of SQL chose the keyword `SELECT` for the relational algrebra π; but you will get used to that very quickly.

Below are a few additional things we can do with a select-from-where, to get us started.

Cartesian product

We can get the Cartesian product of two or more tables by puttting them in a comma-separated list in the FROM clause:

```
csc343h-prof=> SELECT name, sid, grade
csc343h-prof-> FROM Course, Offering, Took
csc343h-prof-> WHERE Course.dept = 'CSC';
           name            |  sid  | grade
---------------------------+-------+-------
 Intro to Databases        | 99132 |    79
 Intro to Databases        | 99132 |    79
 Intro to Databases        | 99132 |    79
 Intro to Databases        | 99132 |    79
 ... etc.
```

Of course, this will create many nonsensical combinations. We will learn all about natural joins and other kinds of join shortly. In the meanwhile, we can use `WHERE` to filter out those nonsensical combinations:

```
csc343h-prof=> SELECT name, sid, grade
csc343h-prof-> FROM Course, Offering, Took
csc343h-prof-> WHERE Course.dept = Offering.dept and   -- New!
csc343h-prof->    Course.cnum = Offering.cnum and      -- New!
csc343h-prof->    Offering.oid = Took.oid and          -- New!
csc343h-prof->    Course.dept = 'CSC';
        name        |  sid  | grade
--------------------+-------+-------
 Intro to Databases | 99132 |    79
 Intro to Databases | 98000 |    82
 Software Design    | 98000 |    89
 Software Design    | 98000 |    72
 ... etc.
```

Compound conditions in a WHERE clause

We just saw our first compound condition. We can build boolean expressions using:

- logical operators: `=, <, >, <=, >=, !=` and `<>`. The operators `!=` and `<>` mean "not equals" in SQL.
- boolean operators: `AND`, `OR`, and `NOT`
- and many other boolean functions and operators. See [chapter 9 (https://www.postgresql.org/docs/9.1/static/functions.html)](https://www.postgresql.org/docs/9.1/static/functions.html) of the PostgreSQL documentation.

Temporarily renaming a table

Similar to ρ in relational algebra, in SQL we can rename a table for the duration of a query. Here we rename `Employee` to `e` and `Deparment` to `d`:

```
csc343h-prof=> SELECT e.name, d.name
csc343h-prof-> FROM Employee e, Department d
csc343h-prof-> WHERE d.name = 'marketing' and e.name = 'Horton';
```

In this case, it makes the query a bit more concise than it would have been:

```
csc343h-prof=> SELECT Employee.name, Department.name
csc343h-prof-> FROM Employee, Department
csc343h-prof-> WHERE Department.name = 'marketing' and Employee.name = 'Horton';
```

but we also know that renaming can be convenient for other reasons, such as distinguishing two occurences of a table in a self-join:

```
csc343h-prof=> SELECT e1.name, e2.name
csc343h-prof-> FROM Employee e1, Employee e2
csc343h-prof-> where e1.salary < e2.salary;
```

```
csc343h-prof=> WHERE e1.salary < e2.salary;
```

Wildcard in the SELECT clause

If we want every column to be included in the result, rather than listing them all, we can use a wildcard:

```
csc343h-prof=> SELECT *
csc343h-prof-> FROM Course
csc343h-prof-> WHERE dept = 'CSC';
 cnum |           name            | dept | breadth
------+---------------------------+------+---------
  343 | Intro to Databases        | CSC  | f
  207 | Software Design           | CSC  | f
  148 | Intro to Comp Sci         | CSC  | f
  263 | Data Struct & Anal        | CSC  | f
  320 | Intro to Visual Computing | CSC  | f
  222 | COBOL programming         | CSC  | f
(6 rows)
```

Naming columns

We can add an AS-expression to choose a column name in the result of a query.

```
csc343h-prof=> SELECT name AS title, dept
csc343h-prof-> FROM Course
csc343h-prof-> WHERE breadth;
       title       | dept
-------------------+------
 Intro Archaeology | ANT
 Narrative         | ENG
 Rhetoric          | ENG
 The Graphic Novel | ENG
 Mediaeval Society | HIS
 Black Freedom     | HIS
(6 rows)
```

Sorting

In relational algebra, a relation is a set and so order doesn't matter. But in the real world, order often does matter. We can sort the results of a query by adding an `ORDER BY` clause to the end of a select-from-where query:

```
csc343h-prof=> SELECT sid, grade
csc343h-prof-> FROM Took
csc343h-prof-> WHERE grade > 90
```

```
csc343h-prof-> ORDER BY grade;
  sid  | grade
-------+-------
   157 |    91
 99999 |    91
 98000 |    92
 98000 |    93
 99999 |    94
 99999 |    96
 98000 |    97
   157 |    98
 98000 |    98
 99132 |    98
 99132 |    99
 99999 |    99
 99999 |    99
   157 |    99
 99999 |   100
(15 rows)
```

The default is to use ascending order, but we can override this by saying `DESC` (not `DESCENDING`):

```
csc343h-prof=> SELECT sid, grade
csc343h-prof-> FROM Took
csc343h-prof-> WHERE grade > 90
csc343h-prof-> ORDER BY grade DESC;
  sid  | grade
-------+-------
 99999 |   100
   157 |    99
 99999 |    99
 99132 |    99
 99999 |    99
   157 |    98
 98000 |    98
 99132 |    98
 98000 |    97
 99999 |    96
 99999 |    94
 98000 |    93
 98000 |    92
   157 |    91
 99999 |    91
(15 rows)
```

We can order according to more than just a single column; we can use the value of an expression to determing ordering. For example, if we had a table including columns called `sales` and `rentals`, we could write `ORDER BY sales + rentals`.

Expressions in a SELECT clause

Instead of a simple attribute name, you can use an expression in a SELECT clause. Here are two simple examples:

```
csc343h-prof=> SELECT sid, grade + 10 AS adjusted
csc343h-prof-> FROM Took;
  sid  | adjusted
-------+----------
 99132 |       89
 99132 |      108
 99132 |       92
 99132 |      109
 ... etc.

csc343h-prof=> SELECT dept || cnum
csc343h-prof-> FROM Course
csc343h-prof-> WHERE cnum < 200;
 ?column?
----------
 CSC148
 EEB150
 ENG110
 CSC100
(4 rows)
```

In the second query, `||` is string concatenation. (See the **PostgreSQL documentation** (https://www.postgresql.org/docs/9.2/static/functions.html) for details on the *many* built-in functions and operators.) Notice that we did not provide a name for the column in the result. In such cases, SQL names it `?column?'. It's essentially saying "I have no idea what to call this!"

Expressions that are a constant

Rather than pull values from a table, a SELECT clause can use a constant value in a column. Here is an example:

```
csc343h-prof=> SELECT name, 'satisfies' as breadthRequirement
csc343h-prof-> FROM Course
csc343h-prof-> WHERE breadth;
       name        | breadthrequirement
-------------------+--------------------
 Intro Archaeology | satisfies
 Narrative         | satisfies
 Rhetoric          | satisfies
 The Graphic Novel | satisfies
 Mediaeval Society | satisfies
```

```
 Black Freedom     | satisfies
(6 rows)
```

We have extracted only courses for which the value of `breadth` is true (which means, in our schema, that the course satisfies the "breadth requirement" of a university degree). And we chose to show this in the result by having the value `satisfies` throughout the second column, which we named `breadthrequirement`.

Case-sensitivity and whitespace

Let's address the lexical level of SQL syntax: how its smallest elements (keywords, operators, and so on) are recognized.

Keywords in SQL are not case-sensitive. Neither are identifiers (names of tables or columns). In addition, line breaks and tabs are ignored by SQL. So, for example, all of these queries are syntactically valid and do the same thing:

```
csc343h-prof=> -- This looks nice.
csc343h-prof=> SELECT sid
csc343h-prof-> FROM took
csc343h-prof-> WHERE grade > 50;

csc343h-prof=> -- This is fine if we just want the answer and aren't saving the query.
csc343h-prof=> select sid from took where grade > 50;

csc343h-prof=> -- This is just ugly.
csc343h-prof=> SELECT SID FROM
csc343h-prof->     TOOK WHERE GRADE
csc343h-prof->     > 50
csc343h-prof->     ;
```

There is no standard convention, but a reasonable one (used in the first query above) is this:

- Put each clause on a line.
- Use all capitals for keywords.
- Use a leading capital for table names.
- Use lowercase for column names.

Typing keywords in all capitals is a little awkward, and you may feel that the queries are shouting at you. It is acceptable in this course to not capitalize keywords. Whatever you do, do it consistently.

SQL is a high-level language

We say that SQL is a high-level language. An SQL query says what we want from the database, not how

to get it. We never have to think about what is stored in memory vs in some level of secondary storage, or what points to what. In other words, we have a level of abstraction that makes it much easier to write queries. In addition, the DBMS can change how the data is stored with no impact on our queries. We say that we have "physical data independence".