CPS221 Lecture: Relational Database Querying and Updating

last revised 10/29/14

Objectives:

1. To introduce the SQL select statement
2. To introduce the SQL insert, update, and delete statements

Materials:

1. library database
2. database to be used in lab (cps221) for on-line demos
3. Projectable of library database schema diagram
4. Sample library query handout
5. Pre-typed queries for cut and paste
6. Database schema diagram for SQL use lab - handout
7. Demo of database for SQL use lab
8. supervises database
9. SQL Syntax handout
10. SQL documentation linked from course web page

I. Introduction

A. We have seen how entities and relationships can be represented by using the relational data model, in which information is stored in tables.

1. Each table has a primary key, which is a set of attributes such that no two rows in the table have the same value.

2. A table may represent either an entity or a relationship.

   a) For an entity, the table includes the entity's primary key and other attributes.

   b) For a relationship, the table includes the primary keys of the entities being related (called foreign keys) plus any attributes of

the relationship itself. The foreign keys, together, constitute the primary key of the table.

B. One of the major strengths of the relational data model is that it supports ad-hoc operations - the ability to access information in the database in a simple way, without having to write a special program to do so. These operations are of two general kinds:

   1. Queries - access information without altering it

   2. Updates - add, delete, or modify information

C. We now have to consider how to actually perform queries and updates. To do this, we will make use of a query language.

   1. Over the years, a number of different query languages have been developed for use with relational databases. They fall into two broad categories:

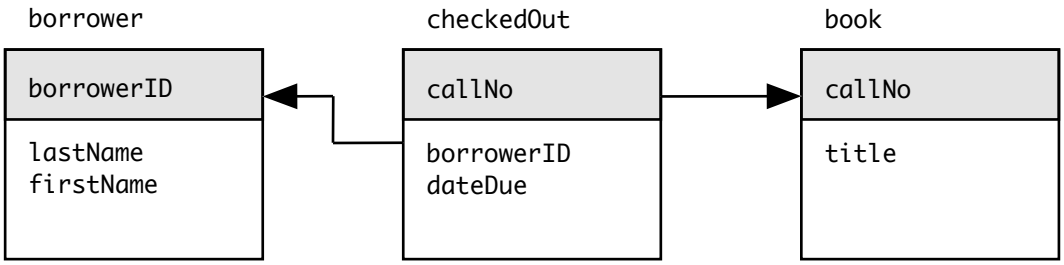
      a) Formal languages that use mathematical notation, and are most useful for theoretical study.

      b) Commercial languages used in actual systems.

      c) We will learn Structured Query Language (SQL) - a commercial query language that has been standardized by ANSI, which is supported by many relational DBMS products, and which is used by Java's JDBC (Java Database Connectivity) facility.

   2. We will also look at SQL facilities for updating a database.

D. For our first examples, we will utilize a simple database for a very small library, realizing the following schema diagram:



PROJECT

(Assume the library is so small that it has at most one copy of any book - hence callNo suffices as the primary key for Book.)

In the case of the checkedOut relation, the foreign keys of the two tables being related are borrowerID, callNo. However, because any given book can only be checked out to one borrower at a time, we use just callNo as the primary key in this case.

DISTRIBUTE QUERY HANDOUT

DEMO:

```
mysql -h dbms.cs.gordon.edu -p
     use library;
     show tables;
```

II. Querying a Database

A. Some simple examples

1. One possible query is one that asks for a particular row of some table.

   Example: "What is the book whose call number is RZ12.905?"

   a) SQL formulation .

```
select *
    from book
       where callNo = 'RZ12.905';
```

   The keyword select is used for all queries, * specifies all columns of the selected row(s), and where specifies the

condition the row(s) must meet. Note also that all SQL queries end with a semicolon, that comparison for equality uses = (not == as in Java) and that strings are enclosed in single quotes. It is good form to put each clause on a separate line, indented with respect to the first line - not required by the language, but facilitates reading.

b) Result (Demo)

```
21873  Fire Hydrants I Have Known Dog    RZ12.905
```

2. It is also possible to formulate a similar query that produces several rows as its result.

   Example: "What books are written by Dog?"

   a) SQL formulation:

```
select *
    from book
    where author = 'Dog';
```

   b) Result (Demo):

```
21873  Fire-hydrants I have known Dog          RZ12.905
34938  21 ways to cook a cat            Dog          LM925.04
```

3. The above queries produce <u>all columns</u> from a one or more rows from the table. Sometimes, we want <u>one or more columns</u> from <u>all rows</u> in a table.

   Example: "List the names of all borrowers"

   a) SQL formulation::

```
select lastName, firstName
    from borrower;
```

   the keyword select is still used, but we explicitly list the

columns we want instead of using *, and we don't have a where clause.

b) Result (Demo):

```
Aardvark            Anthony
Cat             Charlene
Dog             Donna
Fox             Frederick
Gopher          Gertrude
Zebra           Zelda
```

4. The operations of selecting both specific columns and specific rows can be combined in a single query.

   Example: "What is the title of the book whose call number is QA76.093?"

   a) SQL formulation

   ```
   select title
       from book
       where callNo = 'QA76.093';
   ```

   Note that both an explicit column list and a where clause are needed.

   b) Result (Demo)

   ```
   Wenham Zoo Guide
   ```

5. The full power of relational database system comes in when we need to combine information from two or more tables. We will look at a couple of examples now, but will discuss this extensively in a bit.

   Example: When is/are the book(s) Charlene Cat has checked out due?"

   What tables do we need information from in order to answer this question?

ASK

Borrower - since the name only appears there - and CheckedOut - since the date due appears only there. The two tables are related by the common key borrowerID, which is the primary key of one and a foreign key in the other.

a) SQL formulation:

```
select dateDue
      from borrower
        natural join checkedOut
    where lastName = 'Cat' and firstName =
'Charlene';
```

(Note that SQL uses the word and, not && as in Java.)

The "natural join" operation specifies that a new table is to be constructed by taking the Cartesian product of the two tables and then keeping only those rows which agree on their common attribute (called the join attribute): borrowerID. The fact that borrowerID is the join attribute is implicit in the fact that it has the same name in both tables.

b) Result (Demo)

```
2010-10-15
```

6. Joins can be used to combine information from more than two tables.

Example (admittedly a bit contrived):

"What borrowers have books checked out whose author has the same last name as they do?"

What tables are required to answer this?

ASK

All three!

a) SQL formulation:

```
select lastName, firstName
    from borrower
        natural join checkedOut
        natural join book
    where lastName = author;
```

b) Result (Demo)

```
Dog          Donna
Dog          Donna
```

c) Note that the result is to produce two identical rows.. However, there is variant of the SQL select command that eliminates duplicates.

```
select distinct lastName, firstName
    from borrower
        natural join checkedOut
        natural join book
    where lastName = author;
```

d) Demo

B. Review of basic operations

1. Choosing only those rows meeting some criterion

   a) SQL - where clause in select statement

   b) In effect, this operation squeezes a table vertically

2. Choosing only certain columns from a table

   a) SQL - explicit column list (instead of *)

   b) In effect, this operation squeezes a table horizontally. This could result in duplicate rows if we eliminate the column(s) in

which they differ. To avoid this, we must explicitly specify "distinct".

3. Cartesian product

   a) not used in any of the above examples - we will see an example shortly)

   b) SQL - listing multiple tables in from clause

4. Natural join

   a) SQL - connecting tables by natural join in from clause

   b) In effect, this operation does a cartesian join, and then selects only those rows in which columns with the same name from different tables have the same value.

C. For our next examples, we will need to use a more complex database - the same one that you will use in lab.

1. **HANDOUT** of schema diagram

2. Note that we have course id's in several tables, but we store them as two or three separate attributes (department, course_number, and possibly section.) The reason for storing these values is separately is that the relational model requires attributes to be atomic, but we sometimes need to use the different components individually (e.g. we use just department and course_number to link a current_term_course to its catalog information stored in course_offered - the section does not appear in the latter.

3. Discuss primary keys

4. **PROJECT** sample database tables - note correspondence to schema diagram.

D. Additional Query Features

1. Qualified names.

   a) Sometimes, if the same column name occurs in two different tables, it may be necessary to specify from which table you mean for a column to come.

      Example: Suppose we want to print a class schedule for a student with a given id (say 1111111), giving the course id, days, time and room. We need to join the enrolled_in table with the current_term_courses table to get the information we need. However, the following query will <u>not</u> work:

      (switch to cps221 database)

```
select department, course_number, section, days,
        start_time, room
    from enrolled_in natural join current_term_course
    where id = '1111111';
```

      Why?

      ASK

      DEMO - note problem with ambiguity of department, course_number, and section since they appear in both tables

   b) To formulate this query acceptably, we must use:

```
select enrolled_in.department,
       enrolled_in.course_number, enrolled_in.section,
        days, start_time, room
    from enrolled_in natural join current_term_course
    where id = '1111111';
```

(Note that we only need to qualify the otherwise ambiguous columns)

Demo

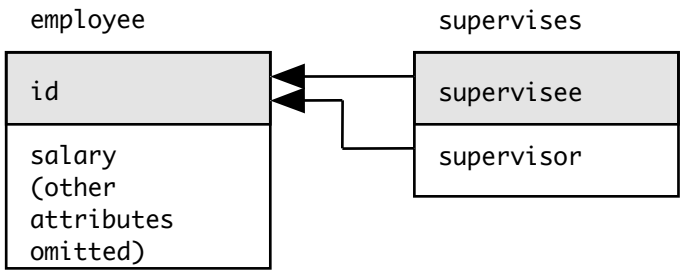
2. Renaming of tables

   a) As the last example illustrated, needing to type the full name of a table over and over when qualifying a name can be unpleasant. To avoid this, it is possible to rename a table within a query. This is illustrated by the following variant of the above:

```
select e.department, e.course_number, e.section,
        days, start_time, room
  from enrolled_in e natural join  current_term_course
  where id = '1111111';
```

   b) In the above example, renaming the table was a convenience. There are times when it becomes an absolute necessity.

   Example: Suppose we had a database that represents the following structure:



   (The names of the attributes of the supervises table denote roles, and are similar in meaning to the use of roles when labelling an association in a UML class diagram. Of course, supervisor and supervisee are actually employee ids)

   Now suppose we want to know what employees make more than their supervisor. This would require joining the employee table with itself (since we need two different salaries) and can be accomplished like this:

```
select e.id
    from employee e, supervises, employee b
    where e.id = supervisee and
          b.id = supervisor and
          e.salary > b.salary;
```

Even if we were willing to type out the full table name every time, we couldn't do the query this way because we are using the same table (with the same name) twice, in two different ways - hence the need to give the table employee table two different names.

c) `use supervises;`
Demo

3. Full Cartesian joins.

a) The above example also illustrates a second point - although the natural join is often what we need, there are times when we want to join tables in some way other than based on equality of values in columns having the same name.

In the above, we needed to join the first usage of employee with supervises based on id = supervisee, while the second usage was joined based on id = supervisor. In neither case are the column names the same.

b) The same issue can arise, even when we don't have to use the same table twice.

Example: in registration database (cps221), in the professor table the department attribute is the department to which the professor belongs, while in the teaches relationship and in the various course tables it is the department that offers the course. Sometimes, a professor teaches a course in a department other than his/her own. Since a natural join between professor and teaches would require that the department attribute in both have the same value (since the column has the same name), a natural join involving these two tables would lose data we might not want to lose.

Suppose want the names of all professors who teach a course in a department other than their own. The query must be

formulated using a full cartesian join, with join conditions explicitly specified:

(1) SQL:

```
select p.professor_name
    from professor p, teaches t
    where p.professor_name = t.professor_name
       and p.department <> t.department;
```

(2) Result - DEMO.

4. Union

Recalling that relations are sets, it is natural to ask whether ordinary set operations are applicable. The answer, of course, is yes. One such operation is set union.

Example: Suppose we wanted a listing of all the courses a given student (say the student with id 1111111) either has taken or is taking. The former are recorded in the course_taken table; the latter in the enrolled_in table.

The schemes of the two tables are not identical, of course, because the former records a term, a number of credits, and a grade - which the latter does not need. (Credits is recorded because the number of credits for a course can be changed, but you still get the number of credits in effect when you took it!) Likewise, the latter records a section code, which the former does not need.

However, appropriate selecting out of column can make the schemes the same - a requirement for set union to be meaningful.

a) SQL (note that the "where" has to appear inside each select). The parentheses are not required, but do enhance readability.

```
(select id, department, course_number
     from enrolled_in
     where id ='1111111')
union
(select id, department, course_number
     from course_taken
     where id = '1111111');
```

b) DEMO

5. Difference

   Another set operation that is useful is set difference. For example, suppose we had a table listing the requirements for a given major. Then we couild find out what courses a given student still needs to take by taking the set difference between the requirements table and the entries for him/her in the courses_taken table.

   Alas, set difference is not supported in mysql, though many sql implementations do support it.

6. Summarization

   One powerful feature of relational databases is the ability to generate summary information easily.

   Example: Suppose we wanted to know how many credits a given student (say id '111111') is taking. This information is available by summing up the credits attribute of the join between the enrolled_in and course_offered tables for that student. A SQL query like the following will do this:

```
select sum(credits)
    from enrolled_in natural join course_offered
    where id = '1111111';
```

   DEMO

7. Grouping

A natural extension of the above is to ask for id and total credits taken for all students.

a) The following query does not work:

```
select id, sum(credits)
    from enrolled_in natural join course_offered;
```

Why? (DON'T DEMO YET)

ASK

We want to sum the credits individually for each student - we don't want the sum for everybody!

b) The following query will work:

```
select id, sum(credits)
    from enrolled_in natural join course_offered
    group by id;
```

DEMO

c) Note: our first attempt is not only wrong, it would actually be rejected by most versions of SQL. (The current version of mysql accepts it, though previous versions did not. It actually shouldn't be accepted!)

d) We could do a more complicated version of the above, in which we print out the student's name instead of the id:

```
select last_name, first_name, sum(credits)
    from student natural join enrolled_in
           natural join course_offered
    group by last_name, first_name;
```

DEMO

8. Outer join

   a) Did you notice that there's actually one row that did not show up in the above?

      ASK

      The row for Emily Elephant did not show up - since she is not enrolled in any courses, the natural join between student and enrolled_in produces an empty set.

   b) Perhaps what we want in this case is a table with a 0 for total credits for any student who is listed in students but is not currently enrolled in any courses. We can get this by using an outer join.

```
select last_name, first_name, sum(credits)
    from student natural left outer join enrolled_in
           natural left outer join course_offered
    group by last_name, first_name;
```

      DEMO - note: most systems would print 0, not null for sum(credits)!

9. Having

   One thing we might want to notice in the above query is what students are taking less than 16 credits. While we could do this by manually scanning the results, its much better to let the DBMS do this for us.

   At first glance, we might think the where clause can be used. Unfortunately, this doesn't work because the where clause selects rows before the groups are formed - whereas we are interested in a particular result of the summary after the groups are formed. The solution is a having clause:

```
select last_name, first_name, sum(credits)
    from student natural left outer join enrolled_in
           natural left outer join course_offered
    group by last_name, first_name
    having sum(credits) < 16;
```

DEMO

10. Order by

a) We have said that the tables in a relational database are sets. One property of a set is that it has not inherent order. Thus, the order of the rows in response to a query is arbitrary. (In fact, most DBMS's produce the results in the order the rows were inserted into the table - but that's not guaranteed.)

b) If we want to force the DBMS to sort the rows into some order before presenting them to us, we can use an order by clause.

DEMO

```
select * from course_offered;
```

```
select * from course_offered
     order by ctitle;
```

c) This facility should only be used when we need it, because sorting is a time-consuming operation.

E. Summary

HANDOUT - SQL Syntax

Go over section on SELECT statements

III. **Updating a Database**

A. The insert statement - handout

B. The delete statement - handout

C. The update statement - handout

D. In addition to these statements, SQL also has statements for maintaining the structure of the database. We will not cover them - you can find them in the online documentation for mysql.

DEMO: mysql documentation linked from course web page