Objectives:

1. To introduce the SQL language
2. To introduce basic SQL DML operations (select, insert, update, delete, commit, rollback)
3. To introduce selected SQL DDL operations (create table, alter, drop, grant, revoke)

Materials:

1. Ability to connect to database / project operations
2. Transparency of architecture of system at Gordon
3. Transparency of example syntax diagram from SQL Reference (connect)
4. Projectable SQL versions of sample queries from last time, plus on-line demo
5. Handout showing commands used to create library example database

I. Introduction

A. Although there are quite a number of commercially-available relational query language, one language has come to be especially important: Structured Query Language. (SQL-pronounced Seequel or SQL)

B. SQL was originally developed for use with IBM's System R - the earliest research implementation of the relational model. In its original form, it was known as SEQUEL (Structured English Query Language). Since then, it has been adopted by many commercial vendors, and has become an ANSI standard - the only query language to be thus standardized - and has undergone a number of revisions - each of which is considerably more complex than its predecessor.

1. The first ANSI standard was 1986. This was revised in 1989 to yield what is now known as SQL 89 - which many commercial products implemented. (The SQL 89 standard is 120 pp.)

2. A more recent standard is SQL 92 (also known as SQL 2). (This standard is 579 pp!) The standard defines three levels of conformance, plus a transitional level between entry and intermediate:

   a) Entry level

   b) Intermediate level
c) Full level

3. The most recent standard is SQL 99 (previously known as SQL 3).

a) This standard is in multiple parts, totaling well over 1000 pages.

b) It incorporates many extensions to more easily support multimedia and object-orientation, so we will discuss it more fully later in the course.

c) There are those who argue that the extensions have resulted in a model that is no longer truly relational - for example, there is a paper I found while researching this topic on the web entitled “Great News, The Relational Data Model is Dead!”, which is basically about SQL 3.

d) Some of the features in SQL 99 were added to their systems by various database vendors in the years between SQL 92 and SQL 99; however, no commercial product fully implements this standard yet.

4. As it turns out, database software from different vendors typically supports slightly different dialects of SQL.

5. Actually, although SQL is based on the relational data model, vendors of database systems based on other models have included a facility for accessing their database using SQL.

C. SQL is both a data definition language (DDL) and a data manipulation language (DML). We will classify statements this way, but the language itself does not draw a distinction between the two types of statement in terms of how they are used. DDL and DML statements can be freely intermixed with one another.

D. SQL can be used in four ways:

1. Interactively.

2. Embedded in an application program, to allow the program to access and or modify the database. Actually, this latter form has two variants:

   a) SQL statements may be embedded in the application program, and processed by a suitably modified compiler or by a pre compiler. (This is called static SQL)
b) SQL statements may be generated as character strings and processed at run time (This is called dynamic SQL).

3. By using JDBC - which resembles dynamic SQL, in that a Java program constructs SQL queries or updates as character strings, and then passes them off to the database to be processed) (It’s actually slightly different in terms of the way it is implemented)

4. Modules consisting of SQL statements can be stored with the tables in the database, to be invoked under various circumstances. (The OO idea of combining state and behavior!)

5. We will focus on interactive SQL for now - embedded SQL will come later in the course (when you do your project) Embedded SQL can use any of the capabilities of interactive SQL, plus there are some statements that are only needed in embedded SQL.

E. The version of SQL we will be studying is that implemented as part of IBM’s DB2 product.

1. This version of SQL implements most (but not all) of the SQL 92 standard, plus many parts of the SQL 99 standard.

2. The following diagram shows the architecture of the way we have installed DB2 here at Gordon. The diagram uses few terms that are used in the IBM documentation in a way that is somewhat differently from the way we used them in our theoretical discussion - specifically, the terms “instance”, “database” and “schema”.

TRANSPARENCY

There are three types of software installations we are using:

(1) The server version of the db2 software is installed on our departmental server machine - jonah. This is also where all the database data physically resides.

(2) Client versions of the db2 software are installed on our 7 other workstations. This allows them to access a database on the server interactively, or to run application programs that access a database on the server. They do not contain any of the actual database data. They do, however, contain a catalog that records information about databases they can connect to (in this case, databases on jonah; but a client’s catalog could actually contain references to databases on many different servers.)
(3) It is also possible for any system that has the db2 JDBC driver installed to access the database via JDBC - for example, this is the case with my laptop. For a JDBC connection, full information about the server must be provided when the connection is made.

a) The software on the server supports any number of instances. Each instance is a totally separate entity, and has no connection to any other instance besides residing on the same system. For this course, we will be using an instance called db2cs352. (The name was totally my choice - I could have called it “aardvark” if I had wanted to!)

b) Each instance contains any number of individual databases. For example, the db2cs352 instance currently contains the sample database used in some of the IBM documentation, a practice database you will be using for homework, and the example library database I will be using later in this lecture. Each database has its own name in the catalog of the instance (sample, practice, library).

c) Each database contains any number of individual schema.

   (1) The major objects in the database (e.g. tables, views) have a two part name of the form SCHEMA.NAME. By default, the schema name is the username of the person who created the object.

   EXAMPLE:

   If I connect to a database using the username “bjork”, and then create a table called “foo”, the full name of the table will be bjork.foo.

   If, however, I connect to the database using the username “aardvark”, and create a table called “foo”, the full name of the table will be aardvark.foo.

   (2) Two objects that belong to different schema may have the same name.

   EXAMPLE: I could create two different tables called “foo” under two different schema names, as described above.
(3) When you reference objects in the database, you can always use their full, two part name. If you only specify an object name, but not a schema name, then a default schema name.

(a) Normally, the default schema name is the name by which you connected to the database.

(b) However, you can use the SET SCHEMA command of SQL to specify a different default schema.

EXAMPLE: The tables comprising the libraryx database were created in the schema demo, so they would have to be accessed either by saying something like demo.borrower or by first issuing the command

```
set schema demo
```

(c) DEMO:

i) Connect to libraryx database on jonah by logging in to a workstation.

ii) Define necessary symbols: (Note the dot by itself at the start of the line!) This sets the path appropriately and performs other setup tasks needed to be able to use the database client installed on the system.

```
. ~db2clien/sqllib/db2profile
```

iii) Connect to the library example database

```
connect to libraryx user bjork
```

(Note: db2 uses linux’s password system to authenticate users)

iv) Show results of

```
select * from demo.borrower
select * from borrower
```

(Note the error this produces)

v) Set the default schema

```
set schema demo
```
vi) Show the effect of specifying the default schema:

```sql
select * from borrower
```

3. DB2 comes with an extensive set of documentation (10’s of 1000’s of pages!). You have been assigned some reading in their *SQL - Getting Started* manual. One manual you will probably want to refer to - though it’s not the easiest reading in the world - is the *SQL Reference* manual.

SHOW LINKS TO BOTH ON WEB SITE

a) One thing you will find in the reference manual is complete syntax diagrams for each SQL statement. (We will present only a small subset in class.)

EXAMPLE: The syntax diagram for the connect statement, used to initially establish a connection to a database:

```sql
TRANSPARENCY
```

```sql
CONNECT

TO server-name
  host-variable

RESET
  (1)

authorization:

USER authorization-name
  host-variable
  USING password
  host-variable

NEW
  password
  CONFIRM password
  host-variable

lock-block:

IN SHARE MODE
IN EXCLUSIVE MODE
ON SINGLE NODE
```
b) The notation used in syntax diagrams is discussed on pages 4-6 of the manual - looking this over before working through the diagrams in the manual is a good idea!

(Note: These are the page numbers appearing at the foot of the page. When reading the manual using Adobe Acrobat Reader, it will give its notion of physical page number, based on numbering the very first physical page in the document “1” (i.e. not recognizing the separate numbering for the table of contents.)

c) The select statement alone is the subject of a full chapter in the manual (chapter 5 - queries) which is 60 pages long! This is because its syntax is broken into portions (subselect, fullselect, and select statement) - some of which can appear in other statements as well!

F. In addition to DB2, we also have an open-source produce known as mysql available. The latter is, in some respects, nicer to use than DB2: it has a nicer interactive interface, and its syntax includes an explicit natural join operator. However, because it does not support transactions, we will not be using it extensively in this course.

II. SQL Queries

A. Probably the most fundamental DML concept in SQL is that of a query.

1. The most frequently used SQL statement is the SELECT statement: a statement intended to get information out of the database. A SELECT statement is basically a query, possibly with some additional components.

2. Queries can also be embedded in certain other statements, as we shall see later.

3. A basic query has the following form:

   SELECT column(s)
   FROM table(s)
   [ WHERE condition(s) ]
   [ GROUP BY column(s)
     [ HAVING condition(s) ]]

   a) Only the SELECT clause and FROM clause are required; the others are optional.
b) The various clauses correspond to relational algebra operations as follows:

\[
\text{SELECT column(s) } \pi \\
\text{FROM table(s) } X \\
[ \text{WHERE condition(s) } ] \quad \square \\
[ \text{GROUP BY column(s) } ] \quad G \\
[ \text{HAVING condition(s) } ]
\]

c) They are executed in the following order

\[
\text{SELECT column(s) } 5 \\
\text{FROM table(s) } 1 \\
[ \text{WHERE condition(s) } ] 2 \\
[ \text{GROUP BY column(s) } ] 3 \\
[ \text{HAVING condition(s) } ] 4
\]

B. A SQL SELECT statement consists of a query, possibly preceded by a WITH clause and possibly followed by an ORDER BY clause. Perhaps the best way to explore this statement is to consider some examples. We will use the same set of queries that we formulated earlier in relational algebra and later in relational calculus. Now, we will formulate them in SQL.

1. The library example database contains exactly the same tables and rows as the example we used in the last series of lectures.

HANDOUT: SQL statements used to create and populate this database

2. Show the tables (just for interest)

\[
\begin{align*}
\text{select * from borrower} \\
\text{select * from book} \\
\text{select * from checked_out} \\
\text{select * from reserve_book} \\
\text{select * from employee}
\end{align*}
\]

C. Work through handout

1. Work through queries, alternating between projected query and actual run on workstation. Stop with “List the names of all employees who earn more than their supervisor.”
2. All of the examples we have done thus far have had a single select clause. When we want to perform set operations such as union, difference, or intersection, we will actually use more than one select clause.

Work through handout examples of union, except, intersect

Note: technically, a select statement consists of one or more fullselects - if more than one, connected by a set operation - optionally followed by an ordered by clause and optionally preceded by a with clause (which we haven’t discussed yet.)

3. Work through remainder of examples in handout - through discussion of implementing natural join and division.

D. Some additional features

1. Various kinds of conditionals that can appear in the WHERE clause. (These are discussed under the heading “predicates” in the manual pages 193-211)

   a) Arithmetic or string comparisons (=, <=>, >, >=, <, <=)

   b) .. BETWEEN .. AND ..

   c) .. LIKE ..

   d) IS NULL

   e) Compound constructs using AND, OR, NOT

2. Functions. These are described beginning on page 216 in the manual. They can be used either in the select list (for projection) or in conditional expressions. Two general types are most commonly used.

   a) Scalar functions that can be applied to individual values - e.g. ABS (absolute value).

   b) Aggregate functions (column functions) that combine information in one column from several rows of a table into a single value - e.g. AVG.

   EXAMPLE: Contrast

   select abs(salary) from employee
with

```
select abs(salary) from employee
```

c) Note that arithmetic operations like +, -, *, / behave like functions that apply to individual values.

E. Subqueries

1. One important capability of SQL is the possibility of embedding a query as a subquery of another query.

2. Consider the following: “List the names and salaries of all employees earning more than the average salary for all employees”

   a) One way to do this would be to issue a select to get the average, and then issue a second select to get the individuals.

   b) SQL allows this to be done in one query by using a subquery

```
select last_name, first_name, salary
from employee
where salary > (select avg(salary) from employee)
```

c) Note that this could also be formulated in relational algebra, though it would be a bit messy!

d) Projectable version / DEMO

3. It is also possible to use a subquery whose result is a set, rather than a single value.

EXAMPLE: “List the names of all borrowers whose last name is the same as that of the author of a book”

```
select last_name
from borrower
where last_name in (select author from book)
```

   a) The subquery (select author from book) forms a set - a list of all the authors.

   b) The “in” predicate occurring in the where clause then checks to see if the borrower’s last name is in this set.
c) This one would be hard to formulate in relational algebra. (You could do so using a theta join - but that’s really a rather inefficient way to go actually go at computing it.)

d) Projectable version / DEMO

4. It turns out that we can also use quantified predicates with sets created by a subquery. To illustrate this, we will have to use a different database - the “sample” database discussed in the *Getting Started with SQL* manual.

```
connect to sample user bjork
set schema db2cs352
select * from employee (widen screen to show)
```

5. “Print the name and salary of any employee earning more than all the employees in department E11”

```
select firstnme, midinit, lastname, salary
   from employee
   where salary > all (select salary
   from employee
   where workdept = 'E11')
```

Projectable version / DEMO

6. “Print the name and salary of any employee earning more than some employee in department A00”

```
select firstnme, midinit, lastname, salary
   from employee
   where salary > any (select salary
   from employee
   where workdept = 'A00')
```

Projectable version / DEMO
III. DML Statements for Modifying the Database

A. Start up db2 with the +c option before doing any of the following (will explain why later)

B. INSERT

Three forms

1. Simplest form: insert explicit values into all columns

   insert into borrower values ('98765','raccoon', 'ralph')

   select * from borrower

   (Note that values are matched with columns positionally - first value goes with first column etc.)

2. It is possible to explicitly specify column names if one is unsure of actual order of columns

   insert into borrower (first_name,last_name, borrower_id) values ('ursula', 'unicorn', '87654')

   select * from borrower

   This form of insert can also be used if one does not have values for all the columns (and the column allows null values)

   insert into borrower (last_name, borrower_id) values ('xerus', '55555')

   select * from borrower

3. Finally, it is possible to embed a select into an insert to copy information.

   Example: suppose we want to make all of our employees eligible to be borrowers if they are not currently such
insert into borrower
    select right(ssn, 4), last_name, first_name
    from employee
    where not (last_name, first_name) in
        (select last_name, first_name from borrower)

(Note: this is a pretty poor way to generate borrower id's - but it illustrates the point!)

C. UPDATE

1. General form: update table set (column = value) where condition

2. Example: Give all employees supervised by aardvark a 10% raise:

   select * from employee

   update employee
       set salary = salary * 1.1
       where supervisor_ssn =
           (select ssn
                from employee
                where last_name = 'aardvark')

   select * from employee

D. DELETE

1. General form: delete from table where condition

2. Example:

   Delete the borrower entry for raccoon:

   delete from borrower where last_name = 'raccoon'

   select * from borrower

3. What would happen if we did a delete without specifying a condition? (no where clause)

   ASK
delete from employee

select * from employee

E. COMMIT and ROLLBACK

1. It appears that - at this point - we have mangled the database. But we really haven't.

   Issue the following commands:

   rollback

   select * from employee
   select * from borrower

2. A very important concept in SQL is the notion of a TRANSACTION

   a) We will discuss this concept more later in the course. For now, we will think of a transaction as a unit of work such that either all the operations in it must succeed or all must fail.

   b) A transaction is normally terminated by entering one of the following statements:

      commit

      or

      rollback

   c) The former causes all changes to the database made during the transaction to become permanent; the latter undoes all of them.

      (Note: until a transaction is committed or rolled back, its effects will NOT be visible to other users of the database, and other users may be locked out from some accesses to the data items involved.)

   d) The system starts an initial transaction when the connection to the database is first made; and starts a new transaction when one is either committed or rolled back. If execution terminates for any reason (user specified or crash) with some transaction still in process, it is automatically rolled back.
DEMO:

Terminate session
Start new session (with +c flag)
Insert a new row into borrowers
Show it's there with select *
Terminate session without committing
Start a new session
Do select * to show it's no longer there

e) Suppose someone does two hours’ worth of data, then terminates their session without committing. What happens?

ASK

Because this is generally not a good thing, most interactive command processors include an automatic commit mode, in which each command typed by the user is automatically committed - which is generally appropriate for interactive input.

In db2 SQL, this is the default mode of operation for the command line processor. For the last couple of demonstrations, I had to disable it. That's what the +c on the command line did:

```
db2
```
- start DB2 with automatic commit enabled

```
db2 +c
```
- start DB2 with automatic commit disabled

IV. Basic DDL Statements

A. The set of DDL statements available in SQL is rich, and we will only introduce them briefly here. Three that are commonly used are:

```
create ...   - to create a new object (e.g. a schema, table, or view - among others.)
```

```
alter ...   - to modify an existing object (e.g. a table or view)
```

```
drop ...   - to delete an existing object (e.g. a table or view)
```

(Note the difference between delete from sometable - which deletes all the rows from a table (leaving behind an empty table), and drop sometable - which drops the entire table (including its data but also its scheme)
B. For now, the only statements you need to know about are

    create table
    and

    create view

1. Go over create table statements used to create library sample database.

   a) Note specification of primary key and foreign key constraints - we will discuss these and other constraints more when we get to database integrity.

   b) Each column in the table must have an appropriate data type. Pages 168-169 of the text list some of the more important data types available (not an exhaustive set).

TRANSPARENCY

   Note the syntax for specifying dates, times, and datetimes when inserting or comparing values. Note, too, that the syntax used for input is not the same as the way SQL displays the values by default!

2. The create view statement has the following basic syntax

    create view viewname as query

   a) EXAMPLE

       Create a view called books_out that lists the titles of all books that are checked out:

       create view books_out as
           select title from book join checked_out on
               book.call_number = checked_out.call_number

       DEMO - WITH AUTOCOMMIT DISABLED

       creating this view, then using select * from it.

   b) Note that creating a view does not store new data in the database. Rather, a reference to the view is handled by "running" the defining query. Any changes in the underlying tables will therefore be reflected automatically the next time the view is accessed.
DEMO: drop a checkout, then repeat select * from view.

c) Note that the action of altering the database scheme with a DDL statement is also under transaction control!

DEMO: rollback and the attempt a select * from the view.

C. We will not discuss the syntax of alter or drop now. They are documented in the *SQL Reference Manual*. 