# CS352 Lecture - SQL

last revised September 5, 2008

*Objectives:*

1. To provide background on the SQL language
2. To review/expand upon basic SQL DML operations (select, insert, update, delete, commit, rollback), with added coverage of subqueries, joins, recursive queries
3. To introduce selected SQL DDL operations (create table, view)

*Materials:*

1. Ability to connect to database / project operations; file with queries
2. Projectable of architecture of db2 system at Gordon
3. Projectable of example syntax diagram from SQL Reference (connect)
4. Handout showing commands used to create library example database
5. Projectable of SQL data types - book pp 77, 121

## I. Introduction

A. As you know, 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 S Q L)   This lecture will serve to provide a bit of background, and also to introduce some key features of SQL that you have not seen yet.

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. A newer standard is SQL:1999 (also 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:1999; however, no commercial product fully implements this standard (or the newer SQL:2003) yet.
4. An even newer standard is SQL:2003. Much of it consists of minor changes to the previous standard (some have called it a "bug-fix"); the major addition is support for XML. I downloaded a draft which consists of 14 pdf files - one of which, alone, has over 1000 pages! The "offical" version is not freely available.
5. The latest standard is SQL:2006, which extends XML support.
6. As it turns out, database software from different vendors typically supports slightly different *dialects* of SQL.
7. 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 three 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). (You have had some exposure to this using JDBC)

3. 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!)

4. We will focus on interactive SQL for now - static SQL will come later in the course (when you do your project)

   a) Static SQL can use any of the capabilities of interactive SQL, plus there are some statements that are only needed in embedded SQL

   b) We will not deal with dynamic SQL at all in this course - you had some exposure to it by way of JDBC in CS211.

   c) We will not deal with SQL modules in this course - that's an advanced topic beyond the scope of CS352.

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.

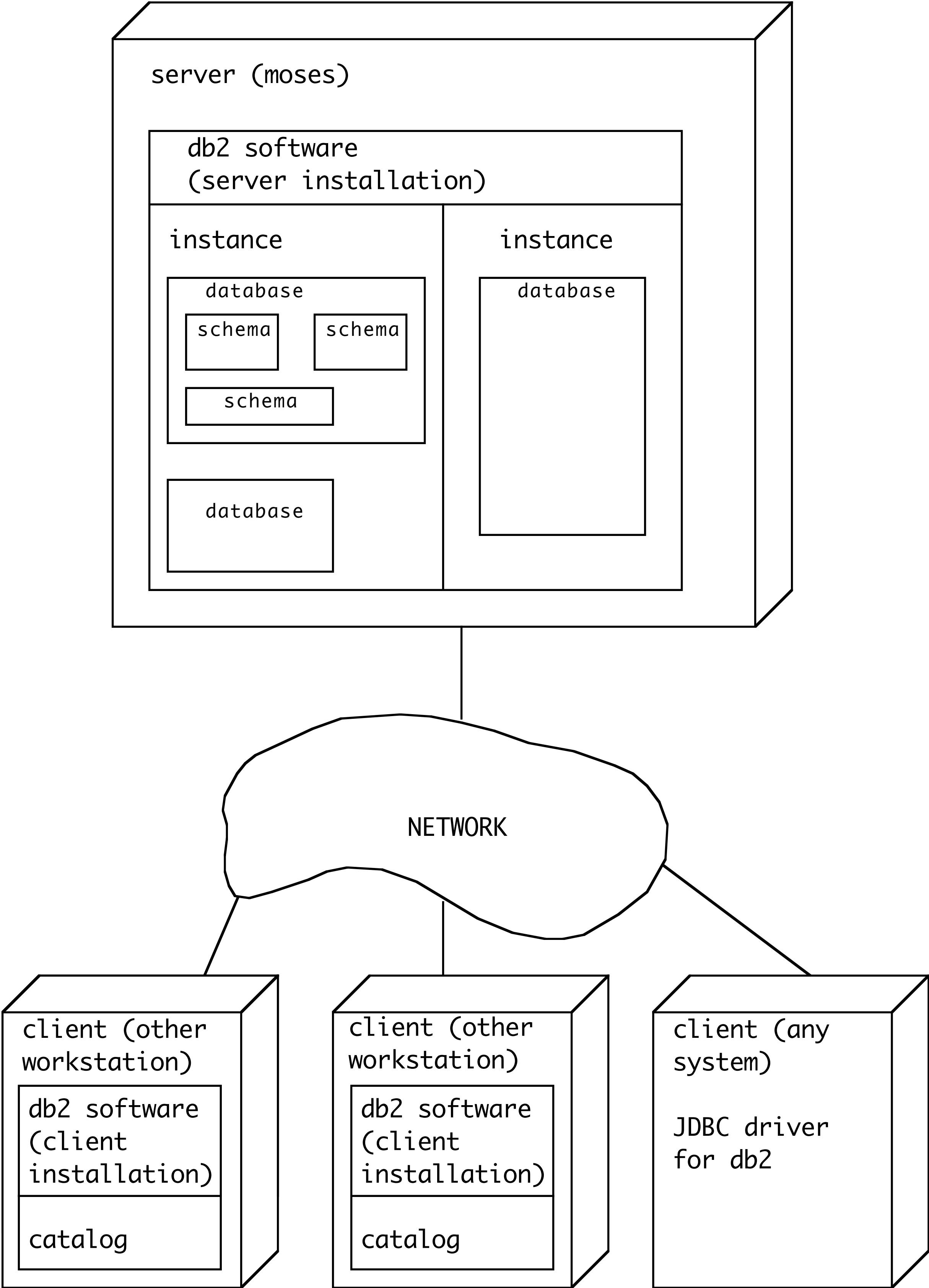
2. The following diagram shows the architecture of the way we have installed DB2 here at Gordon. The diagram uses few terms that are used in the IBM documentation in a way that is somewhat differently from the way we used them in our theoretical discussion - specifically, the terms "instance", "database"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

   *PROJECT*

   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 - moses. This is also where all the database data physically resides.

   (2) Client versions of the db2 software are installed on our 8 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 <u>not</u> 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 moses; but a client's catalog could actually contain references to databases on many different servers.)


server (moses)
db2 software
(server installation)
instance
instance
database
schema
schema
schema
database
database
NETWORK
client (other
workstation)
db2 software
(client
installation)
catalog
client (other
workstation)
db2 software
(client
installation)
catalog
client (any
system)
JDBC driver
for db2

(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. The JDBC driver is written in Java, and hence runs on any system that runs Java - it does not have to be running db2. 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 <u>instances.</u> 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 `db2inst1`. (This is the default name when doing a server installation - I could have called it "aardvark" if I had wanted to!)

b) Each instance contains any number of individual <u>databases.</u> For example, the db2inst1 instance currently contains the sample database you are using for homework, a database you will use for the design project, and the example library database I will be using later. Each database has its own name in the catalog of the instance (`sample, design, library`).

c) As we have already noted in conjunction with the homework, each database contains any number of individual <u>schema.</u>

(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, as you have been doing for the homework.

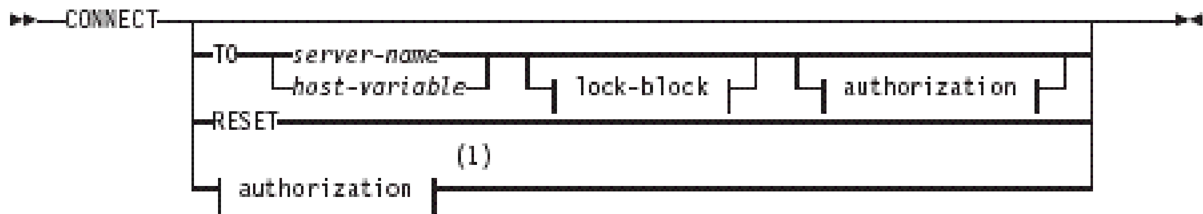
3. DB2 comes with an extensive set of documentation (10's of 1000's of pages!).

   SHOW MANUAL ON BLACKBOARD SITE

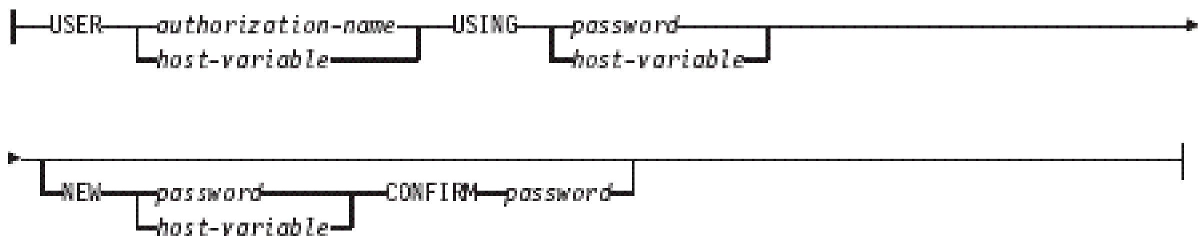
   a) One thing you will find in the reference manual is complete syntax diagrams for each SQL statement.

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

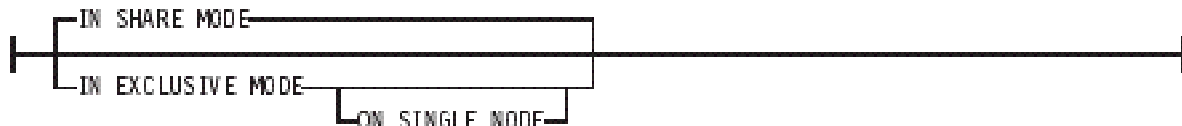
      *PROJECT*




authorization:




lock-block:



b) The notation used in syntax diagrams is discussed on pages xiii-xv 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 introductory material.)

c) The select statement alone is the subject of a full chapter in the manual (chapter 4 - queries) which is 54 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. Also accessible on the Blackboard site is a book by Graeme Birchall called DB2 UDB Cookbook - which is a SQL book specifically based on the DB2 implementation of SQL. You will find this very helpful as you do homework throughout the course.

SHOW

G. In addition to DB2, we also have an open-source product 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. You were introduced to queries in CS211, and will get lots of practice with various sorts of queries in the homework, and both the text and Birchall books cover them extensively; therefore, I will not spend time on them here. However, I do want to spend some time on three feature that are powerful, but potentially tricky.

B. 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);
   ```

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

   c) Projectable version - connect to library / schema 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 actually go at computing it.)

   d) Projectable version - connect to library / schema demo

4. We can also use quantified predicates with sets created by a subquery. “Print the name and salary of any employee earning more than all the employees in department E11” (Using employee in sample)

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

   Projectable version - connect to sample/set schema db2inst1

5. “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

C. Various kinds of Joins

1. In our study of relational algebra, we learned about a variety of join operations:

   a) The full cartesian join - represented by X - e.g.

      A X B

      means each row of A is paired with each row of B

      How big will the resulting table be?

      ASK

      This results in a table whose number of rows is (number of rows in A) * (number of rows in B).

   b) The natural join - represented by |X| - e.g.

A |X| B

means each row of A is paired with those rows of B which have the same values for all the columns they have in common - e.g. if both A and B have a `callno` column, then rows of A are paired only with rows of B having the same value in the `callno` column.

How big will the resulting table be?

ASK

We can't know for sure. It could be empty (if no row in A matches a row in B). It could be the same size as the cartesian join (if there are no columns with the same names, or all the rows have the exact same value in the relevant column).

What if the join column(s) include the primary key for one of the tables? (Say A)

ASK

In this case, the result will be no bigger than number of rows in the other table (e.g. B). The reason for this is that each row in B will jon with at most one row in A.

c) The theta join - represented by $X_\vartheta$ - which is like the cartesian join, except that some test is applied and only joined rows passing that test are kept.

d) The outer join - which comes in three varieties (left, right, and full), written as ⊐X|, |X⊏, and ⊐X⊏. - e.g.

A ⊐X| B

means that we do a natural join between A and B, but then any row of A that does not join with some row of B is paired with a "manufactured" row of B (all attributes null) so no rows of A are lost.

2. SQL has forms that are equivalent to each of these

a) The cartesian join is specified by simply listing the names of the tables to be joined, separated by commas - e.g.

```
SELECT something FROM A, B
```

is equivalent to A X B

and

```
SELECT something FROM A, B, C, D
```

is equivalent to A X B X C X D

b) The natural join is specified by using the phrase natural join - e.g.

```
SELECT something FROM A natural join B
```

is similar to A |X| B

(1) However, it differs in one important respect. Any idea?

ASK

|X| is defined in relational algebra as keeping only one copy of common columns (i.e. there is an implicit project). In SQL, one would have to specify this explicitly

(2) Unfortunately, not all dialects of SQL support natural join. In particular, DB2 does not. One achieves the effect of natural join by either using the WHERE clause or the next form of join we will consider (which is equivalent to the theta join)

Example: Consider the tables

```
book(callno, title)
checked_out(callno, borrower_id, due)
```

The relational algebra book |X| checked_out could be expressed in SQL by:

```
SELECT book.callno, title, borrower_id, due
      FROM book, checked_out
      WHERE book.callno = checked_out.callno
```

(3) Note the need for the use of qualified names where both tables have a column name in common. Typing is often reduced by using correlation names - e.g.

```
SELECT b.callno, title, borrower_id, due
     FROM book b, checked_out c
     WHERE b.callno = c.callno
```

c) The theta join is expressed by using the phrase join .. on - e.g.

```
SELECT author
   FROM book JOIN borrower ON author = name
```

is equivalent to the relational algebra

book X borrower
 ϑ author = name

(1) One important use of join .. on is to produce the equivalent of a natural join in implementations that lack the natural join keyword:

Example: book |X| checked could also be written

```
SELECT b.callno, title, borrower_id, due
     FROM book b JOIN checked_out c
          ON b.callno = c.callno
```

(For natural join, this is preferable to using where, since on is part of the join operation, selection done after the join).

(2) It is important to remember, though, that join on need not involve a natural join - the join condition can be any join condition.

d) The outer join is specified by using the phrase left outer, right outer, or full outer before the word join.

(1) Example: include the borrower id's of all the borrowers, even if they don't have a book out:

```
SELECT b.callno, title, borrower_id, due
     FROM book b LEFT OUTER JOIN checked_out c
          ON b.callno = c.callno
```

(2) The condition in the join is often equivalent to a natural join - though it need not be In fact, SQL implementations that do support the phrase natural join typically also allow outer joins to be combined with it (yielding a meaning similar to relational algebra operations) - e.g.

A ⊐X| B

would be written ... A natural left outer join B

(3) By way of contrast, joins that are not outer joins are somtimes called inner joins, and the word inner can be explicitly used, though it is not required

e.g. A join B on ... could be written A inner join B on ...

D. Recursive Queries

1. An interesting kind of problem arises when it is necessary to perform a recursive query in SQL.

a) For example, consider a table listing a person's name and the name of his/her parent - e.g.

```
person(name, parent)
```

For simplicity we will use just first names, and each row will record just one parent (so a given person might appear in the table twice - one row for each parent).

(1) It is easy to print the names of all the children of a given parent:

```
select name
    from person
    where parent = ____
```

(2) A more complicated case arises if we want all of a given parent's grandchildren. In this case, we need to join the table with itself, and make use of the rename operation:

```
select p.name
     from person p join person q
       on p.parent = q.name
     where q.parent = ___
```

We could, of course, use a similar approach to list someone's great-grandchildren (in which case the table is joined with itself twice)

b) Now consider the following more challenging problem: print the names of all of a given person’s descendants, regardless of how many generations are involved. This is, of course, a recursive query, based on the following recursive definition.

A is a descendant of B if B is A's parent, or A's parent is a descendant of B

2. At one point, it was impossible to formulate a query like this in SQL, because SQL does not provide for recursion. (Instead, one would need to embed a SQL query in a host language that did support recursion, or use some other query language like Datalog).

3. Recent versions of SQL provide for handling case like this by using a union between a base table and a recursive table to create a temporary table using a with clause.

   For example, here is a SQL formulation of this query

```
with descendant(name) as
  (  select name
          from person
          where parent = _____
   union all
     select person.name
          from person, descendant
          where person.parent = descendant.name
  )
select *
     from descendant;
```

   Two things are going on here

   a) The with clause is used to create a temporary table. A temporary table is one that exists just for the duration of the query. Thus, in the with clause, we have to specify the table's name, its scheme, and an "as" clause that defines its content. (In this case, it is a table containing all the descendants of a given person, and has just a single column, though it could have multiple columns if needed). A "select *" is used to print this table out. [ Note: use of join .. on in this case is forbidden in DB2 SQL - hence the use of where ]

   b) The content of the table is defined by using a "union all" between the base case and the recursive case of the definition

   c) Demonstration (in database genesis under schema bjork)

```
select * from person;
```

```
select name
    from person
    where parent = 'adam';
```

```
select p.name
   from person p join person q
     on p.parent = q.name
   where q.parent = 'lamech';

with descendant(name) as
  (     select name
        from person
        where parent = 'adam'
    union all
   select person.name
        from person, descendant
        where person.parent = descendant.name
  )
select *
   from descendant;
```

(Note: "on" cannot be used because we're doing a union, not a join)

## 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) 

Connect to library/schema demo

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');
```

(This fails because first_name was declared not null)

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. As we noted at the start of the course, 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 77 and 121 of the text list some of the more important data types available (not an exhaustive set).

   PROJECT

   Note the syntax for specifying dates, times, and datetimes when inserting or comparing values. Note, too, that the syntax used for input is not he 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*.