CPS352 Lecture - Course Introduction; Fundamental DBMS Concepts

Last revised July 24 , 2008

Objectives:

1. To introduce the syllabus and key issues of the course - showing how they are related.
2. To contrast file-processing systems and DBMS-based systems.
3. To introduce key DBMS concepts and terminology

Materials:

1. Course syllabus
2. Example database (intro); handout of SQL used for it [test run examples on it]

Course Introduction

Distribute, go over syllabus. Mention that we will be relating key ideas to the course schedule as we go through this lecture.

I. File Processing Systems vs Database Management Systems

A. At the outset of our course on database management systems, we want to carefully distinguish two different approaches to meeting the information processing needs of an organization: a file-processing approach versus a database management system approach.

1. Historically, these two approaches evolved successively.

   a) Early computer applications were always developed using the file processing approach, because that was the only approach known.

   b) The DBMS approach was developed in the 1960's, and has come to be used in a variety of application areas - many falling into the broad category of "business data processing", but for other areas as well.

2. Though we will distinguish these approaches, we don't want to treat them as totally mutually exclusive. In fact, for many applications, the file processing approach is the best way to go. (Word processing and spreadsheets, for example, work this way) Both approaches can be used at various points in the information processing activities of an organization; each is legitimate in its own proper domain.

B. Regardless of which approach is used, an organization's needs are going to be met by a collection of files containing its operational data and programs that manipulate this data.

1. We distinguish between operational data that is relatively permanent and transient input/output data.

   Example: In a payroll application, the operational data includes the basic employee data such as name, SSN, rate of pay etc., plus year-to-date totals for income, taxes etc. This data is retained in storage permanently.

   (This is distinct from transient data like weekly time card information, which is discarded once processed.)

2. In either case, we call the collection of files that together contain the organization's operational data its DATABASE.

3. What distinguishes the file processing approach from the DBMS approach is how the programs relate to the operational data. (A file processing approach is often still used with transactional data.)

C. We might say that a file-processing approach is characterized by a close relationship between programs and data.

1. Each program is written to process a certain file or group of files and must embody detailed knowledge about the structure of each file it uses.

   Example: Consider a program that prints mailing labels to send a mailing to each customer on an organization's customer list. This program presumably works with a customer file, and must, therefore, incorporate knowledge about how the customer file is structured.

   a) Suppose that the customer file consists of one record per customer, with each record containing customer name, address, city, state, ZIP, plus the name of the salesman handling the account, the total sales to the customer this year to date, and a flag indicating what kind of mailings the customer might receive.

   b) Each program accessing this file would have to incorporate this information in some language-appropriate way - e.g. as a C/C++ struct declaration:

```
struct
{
        char[30] name;
        char[30] address;
        char[20] city;
        char[2]  state;
        int      zip;
        char[20] salesman;
        float    ytd_sales;
        char     mail_code;
} Customer;
```

2. As a corollary, any change in the structure of the file will necessitate a change in the program.

   Example: Suppose the company decides to adopt a policy of giving different discounts to different customers. This might call for adding a discount field to the customer file. Of course, this change does not affect the mailing list directly; nonetheless, the mailing list program must be modified to reflect the changed file structure.

3. To avoid this unintended coupling between unrelated programs, it is common to design file-processing type systems so that each application area "owns" its own files.

   Example: We might maintain separate customer files for the mailing list application and the billing applications. The former would include only customer name, address, city, state, ZIP, and mailing code. The latter would certainly include name, salesman, YTD sales, and the discount. It might also include the address information so that the bills can be mailed. But it would not include the mail list code.

4. File-processing based systems, then, tend to be characterized by a proliferation of application-specific files, each with its own format. Certain data items are stored redundantly - i.e. in more than one place in the database. This, however, creates new problems:

   ASK CLASS

   a) Wasted storage (becoming less of a problem as storage costs go down, but still a concern, especially when one thinks of backup using a network.)

b) Update problems: when an item of information has to be changed, it may need to be changed in several different places in the database. This means extra work each time an update has to be done.

c) Inconsistency problems: over time, it is possible that the database may contain two different values for the same data item in two different places, because some update operation did not catch all of the places that need to be changed. This causes confusion.

Example: Gordon's first computerized registration system maintained a separate student file for each academic term. Each file contained various personal data on the student, the name of his advisor, and a list of the courses he/she was enrolled in that term. The file also contained space to record the grades for each course taken, though of course these slots would not be filled in until after the end of the term.

(1) As registration time for a new term approached, the computer center would copy data from the current term's file into a file for the new term, blanking out the list of courses registered for but leaving all else intact.

(2) At some point in time, the registrar's office could have three different files active:

(a) The term just completed, awaiting the posting of grades and printing of grade reports, plus the possibility of grade changes by the professor.

(b) The current term.

(c) The upcoming term, since registration for a new term is held about five weeks into the preceding term.

(3) Any change in basic student information would have to be posted to ALL the active files. Sometimes, this would not be done.

(a) In one case, a student changed into the computer science major in mid-term, and I was assigned as her advisor. This was duly recorded in the current term's file; however, the file for the new term had already been created, containing her old advisor's name, and this was not changed.

(b) As a result, I got her grade report for one term, but the next term the grade report was sent to her old advisor. We caught this, and the file was updated; but not before the outdated information had been propagated into yet another term's file.

(c) It took multiple terms before all the records agreed that I was this student's advisor. In one case, she was sent back from registration because my signature was on her card and the computer said someone else was her advisor (a year after she had changed majors)!

d) Data isolation problems: it is not easy in such a system to pull together a report containing all the information stored on one particular entity, since it is scattered over many files, each with a distinctive format.

e) Atomicity problems: Suppose a bank customer uses an ATM to transfer $100 from his/her checking account to savings. This involves a program debiting the checking account for $100 and crediting the savings account with $100. Now suppose the bank's computer crashes in the middle of the operation.

(1) If the debit occurs first, then the computer crashes, the customer might be out $100.

(2) OTOH, if the credit is done first, the customer might gain $100 at the bank's expense.

(3) To prevent this kind of problem, we must somehow ensure that the transfer transaction is ATOMIC - either the entire transaction occurs, or nothing occurs. Each program that changes the database must deal with these issues.

f) Concurrency problems: if the data is contained on a multi-user system, it may be that two different users might access the same data item simultaneously.

(1) If both are trying to update it, inconsistencies could result.

Example: in a bank account system, if a customer is depositing money at an ATM at the same time that an EFT program is transferring money from the checking account to savings, the following scenarios may occur. (Assume that the customer has $1000 in his checking account and deposits $200 more at the

same moment the EFT program is transferring $100 from checking to savings):

Scenario 1: ATM program reads $1000 balance
EFT program reads $1000 balance
ATM program adds $200 and writes back the sum ($1200)
EFT program subtracts $100 and writes back the difference ($900)

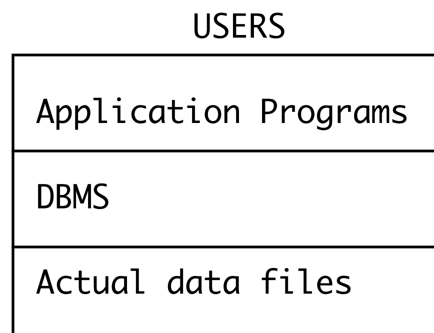
Final balance is $900, not $1100 as it should be. (Customer is mad!)

Scenario 2: ATM program reads $1000 balance
EFT program reads $1000 balance
EFT program subtracts $100 and writes back the difference ($900)
ATM program adds $200 and writes back the sum ($1200)

Final balance is $1200, not $1100 as it should be. (Customer is happy, but bank is mad!)

(2) To prevent such inconsistencies, each program modifying shareable data must be aware of other programs that might access the same item.

D. In contrast, a database management system approach breaks the tight coupling between application programs and data, by putting a software layer in between:



1. Application programs that need data do not get it directly from the files where it is stored, but rather from the DBMS, which in turn gets it from the file. Application programs are not allowed to access the data directly.

2. In addition to the data itself, the database (collection of files) also contains META-DATA: data about the data.

   a) This takes the form of a data dictionary, which contains at least two things for each data item in the database:

      (1) A standard name for the data item which application programs use when they want to access it - e.g.

         Customer.name, Customer.address, Customer.street, Customer.city, Customer.state, Customer.ZIP, Customer.YTD-Sales, Customer.discount-percent, Customer.mail-code.

      (2) Where the data item is stored (what file it is in, and what field in the file), so that the DBMS can locate it.

      (3) DEMO:

         Open a terminal window to moses; widen it

```
db2 -t +p
connect to intro user bjork;
list tables;
describe table accounts;
```

   b) The data dictionary often contains other information as well. For example, it may contain:

      (1) Security constraints: rules as to who is allowed to examine or update a given data item.

         (a) In a file processing system, security must be done on a file by file basis: any user having read/write access to a file has read/write access to all the fields in it

         (b) In a DBMS system, security can be applied item by item. For example, the mailing list application might have access to a customer's name, address, and mailing code, but not his discount percentage - even though all these items may be stored in the same file. The DBMS can enforce these security constraints, since all accesses to the data go through it.

(c) DEMO: In original window from above

```
select * from accounts;
```

Open another terminal window

Start db2 with `-t +p`

```
connect to intro user aardark;
set schema bjork;
select * from accounts;
```

`select * from account_owners` (a view - we will discuss this a bit more later, but notice how we restrict access to "public" columns)

```
update account_owners
   set address = 'MacDonald 212'
   where owner = 'Aardvark';
select * from account_owners;
```

Repeat in original window: `select * from accounts;`

(d) HANDOUT - SQL used to create this database

(2) Integrity constraints: often, the values of certain items in a database are logically constrained to only certain possibilities.

Example: a grade field in the Gordon registration system may only contain A, A-, B+ ... D-, NC, I or W. Any other value (e.g. Z) is meaningless. It is important for software that modifies such an item to ensure that the new value obeys the appropriate constraints.

(a) Under a file-processing approach, this is difficult since each program that accesses the data must know and apply the constraints. The problem becomes especially severe if a new constraint must be added or an existing one altered: every program accessing the data must be modified to the new rules.

(b) Under a DBMS approach, the data dictionary entry for the item can contain constraint information which the DBMS software can check whenever the item is changed, since all changes to the item are done through the DBMS.

(c) DEMO: In the window connected in as bjork:

```
update accounts
   set checking_balance = checking_balance + 100
   where owner = 'Aardvark';
select * from accounts;
update accounts
   set checking_balance = checking_balance - 200;
```

3. The DBMS can ensure atomicity of transactions by using one of several approaches we will consider later in the course.

4. The DBMS can allow multi-user access to the data by managing accesses in such a way as to prevent inconsistencies. For example, an application that reads a data item may be required to tell the DBMS that it intends to update the item, in which case the DBMS will lock that value until the update is complete.

   Thus, in our ATM/EFT scenario, once the ATM application read the checking account balance, that item would be locked until the ATM update was complete, and the EFT application would be made to wait until the lock is released.

   a) In a file-processing system, every program that accesses a shared file needs to be aware of all the other possible accesses to that file. (Or - and more typically - files are locked so that only one program can be modifying a given file at a time.)

   b) A DBMS can manage concurrent access to files.

      DEMO: start two "bjork" connections to intro using db2 `t +p +c`

      Consider the following series of operations, which might be used to effect a transfer of money from one account to another. Clearly, we don't want someone else to be able to see the balances between these two operations, lest he/she mistakenly believe that 'Aardvark' has $100 more than he really does

```
update accounts
   set checking_balance = checking_balance + 100
   where owner = 'Aardvark';
update accounts
   set savings_balance = savings_balance - 100
   where owner = 'Aardvark';
```

Issue the first `update` from one window, then try `select * from accounts;` from the other. Note how it is blocked. Now finish the transaction and `commit` it - note how the access attempt can now "see" the updated balances

(Note: normally db2 treats each statement as a transaction; issuing `+c` at startup caused it to require an explicit `commit` to end a transaction.

5. DBMS's also often make it easier for users to get at the data in an ad hoc way.

   a) Under a file processing approach, any access to data requires a program to be written for that purpose.

   For example, the get a report of total YTD sales for all customers receiving the "F" mailings, a program would have to be written containing:

   - The definition of the record layout.
   - Code to open and close the file.
   - A loop like the following:

   ```
   sum = 0.0;
   for each record in the file
      if (mail_code == 'F')
        sum += ytd-sales;
   ```

   If no program has been written to generate a given type of report, then someone who needs that type of report must either do without or be willing to have a programmer paid to write it (and be willing to wait until he/she can finish the program!)

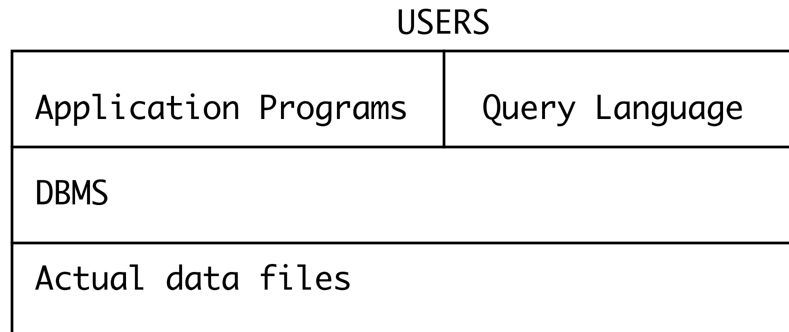
   b) Most DBMS's also include a QUERY LANGUAGE which allows a moderately sophisticated user to get at information in the data base directly, without going through an application program.

   Example: A DBMS that supports the SQL query language would allow an interactive user to get an answer to the above question by typing a query like:

   ```
   select sum(ytd_sales)
          from customer
          where mail_code = 'F';
   ```

(1) Such queries are possible because the data dictionary is able to provide a translation between item names such as ytd_sales and mail_code and actual physical locations in the database.

(2) Thus, our picture becomes:



Thus, our DBMS has two interfaces: one for application programs (which may call the DBMS using the regular procedure call mechanism of the language they are written in), and one for direct access by end users, using a query language.

(In fact, some microcomputer DBMS's have only the latter interface.)

6. We have seen that putting a DBMS software layer between the data and users of the data has many advantages in terms of eliminating redundancy and inconsistency while facilitating security, integrity, multi-user access and end-user queries. However, there is a price tag on this: the additional layer of software can result in a performance penalty:

   a) At least, there is the additional processing overhead each application incurs by going through a software layer to get at the data it needs, rather than getting it directly.

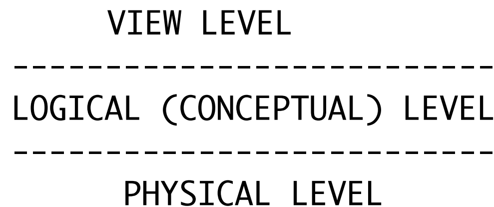
   b) If the application software "knows" how the data is stored physically, it may be able to arrange its accesses to the data in an optimum way in terms of processing efficiency. The DBMS level deprives the application software of this knowledge.

   c) Sophistication of DBMS design, coupled with increasing speeds of computer hardware, now typically allow the benefits of a DBMS without penalizing performance in an observable way, though the answers on this are far from all being in. (More on this later in the course.)

II. The Structure of Database Management Systems

A. Actually, our model of a DBMS coming between users/applications and the data is somewhat simplified from the way real systems work.

1. A DBMS provides three different ways of looking at the data - three levels of abstraction, each of which builds on the level below it:



2. At the physical level, data is stored in files.

   a) Each file has a particular record layout for that data.

   b) Files may have indices or hash structures associated with them to improve efficiency of processing.

   c) One file may contain pointers to another file - i.e. physical addresses on disk of related information in another file.

   Thus, the physical level of data description is much like the way that a file processing approach sees data.

3. The logical level represents the "big picture" of the overall database. At this level, data is thought of in terms of objects and relationships between those objects.

   At the logical level, certain details of the physical storage of the data are abstracted out. For example, at this level no notice is taken of how a particular file is organized (sequential, indexed, or hashed). So if one tries to look up a particular student given his/her id, the DBMS may have to scan the file sequentially (looking for the correct value), or it may lookup the correct record using an index tree, or it may use a hashing function to locate the correct record - the user does not need to know.

4. At the view level, each user of the data (application program or interactive user using a query language) can be presented with the relevant portion of the database in an appropriate way. There is one logical (conceptual) description of the database, but there may be many views.

a) Often, a view is a subset of the information on one particular object.

Example: at Gordon, a great deal of information is stored about each student:

name
home address
parents' names and address
home church
campus address
major(s) and minor if any
advisor
gpa
financial aid allocation
payment plan
details about amounts owed and paid for the current year
chapels attended so far this term

(1) At the physical level, this may be stored in one file or many; but at the logical level this is all thought of as information about one object: a student.

(2) Further, many different offices need to access information about a student:

CSD
Registrar
Admissions and financial aid
Bursar
Chapel
etc.

(3) However, each office needs only a subset of the total data on the student, and SHOULD HAVE ACCESS ONLY TO THAT SUBSET (for privacy and security reasons). The DBMS allows the creation of individual views for each office - so that they can only see a subset of the data appropriate to them - e.g.

(a) Example: CSD view:

name
home address
parents' names and address
home church
campus address
major(s) and minor if any
advisor

(b) Example: Chapel view:

name
advisor
chapels attended so far this term

b) A view may not just be strictly a subset of the logical data on an object. It may also include virtual fields.

Example: The total of current outstanding bills owed by all students is an item of information that is of interest to upper management in the college, as is information about the number of students whose payments are overdue, etc. This kind of information should appear in one of their views of the database. At any time, this figure is computable by combining information (typically multiple items) in all the student records; thus, it need not be represented by a data item at the physical level, but may be computed as needed. However, the view presented to users needing this data may make it look as if such a field exists; but when they try to access it the DBMS converts the access request into the appropriate computation in a way that is transparent to the user.

5. One very important goal of a DBMS is DATA INDEPENDENCE: changes at one level of the database should be possible without in any way affecting higher levels.

a) Physical data independence: changes at the physical level should not affect the logical or view levels.

Example: a decision is made to change a certain physical file from indexed to hashed organization to improve performance. This change should not be visible at the logical or view levels.

b) Logical data independence: changes at the logical level should not affect the view level.

Example: a decision is made to add a new item of information to data stored about an object at the logical level. This must, of course, be reflected at the physical level, but should not affect any views other than those for which the new item of information was added.

B. Data Models

1. In a file-processing environment, there is only one way of describing data: the physical description of how it is stored. (Cf the C/C++ struct declaration above.)

2. However, in a DBMS environment, we have to have several different ways of describing data - at least one for each level. A data model is an abstract way of describing data. It is possible to have:

   a) A physical data model, for describing data at the physical level.

   b) A logical data model, for describing data at the logical (conceptual) level (and also at the view level, since views are really customized variants of the logical level of description.)

   c) We will focus on logical models.

3. The book mentions seven major logical models. Five have been used as the basis for commercial DBMS's; the first one listed is more of an abstract model, and the last is more used for communicating information between systems than for storing it. We will look at all of them over at some point in coming lectures:

   a) The entity-relationship model

   b) The hierarchical model (largely obsolete - will only be looked at briefly)

   c) The network model (largely obsolete - will only be looked at briefly)

   d) The relational model

   e) Various object-oriented models.

   f) The object-relational model (basically a set of extensions to the relational model to provide some OO features).

   g) XML

4. The latter six are listed in historical order of development. Even though the development of these different models spans several decades, all but the ER model (a conceptual model) and XML are currently in use as the basis of commercial systems.

a) One reason for this is the existence of LEGACY SYSTEMS - databases that may have been set up 30 years ago, but which are still in everyday use.

b) Another reason is that the hierarchical and network models tend to be more efficient (but less flexible) than the relational model. (This is much less of an issue today than it once was due to system speeds)

5. Because it is the most widely available/used model today, we will spend most of our time on the relational model, which currently dominates the field.

6. Note relationship to the course syllabus: the first half of the course plus a week after quad break will be spent on the various data models, with the bulk of the time spent on the relational model. The rest of the course will deal with implementation issues, mostly discussed in the context of the relational model, and further applications of database systems.

C. Organization of a Typical DBMS

As pointed out in the book, a DBMS has three major components:

1. A query processor that handles requests to access data coming from users either directly or through an application program.

2. A storage manager that is responsible for physically storing the data on disk., including managing transactions to ensure atomicity and deal with concurrency issues.

3. Note how this relates to the schedule in the syllabus: much of second quad is given over to exploring issues related to these various components of the DBMS.

D. The Transaction Concept

1. A key concept in DBMS design is the notion of a TRANSACTION - a unit of work that has four key properties (sometimes called the ACID properties)

a) It is Atomic - either it happens in its entirely, or nothing of it happens - e.g. if a customer is doing a funds transfer at an ATM and something crashes mid-transaction, both accounts will reflect the “before” balances - it won’t be the case that one has been debited but the other has not been credited or vice-versa.

b) It is Consistent - if the database is consistent before the transaction is started, it is consistent when the transaction is complete. (Though the database may become momentarily inconsistent during the execution of a transaction - e.g. during a transfer of funds transaction at a bank, there will be a moment when the sum total of balances in the customer's account is either too high or too low by the amount of the transfer.)

c) It is Isolated - other transactions do not "see" any momentary inconsistencies introduced the the transaction.

d) It is Durable - once it has been done, its effect is guaranteed to remain, even in the case of hardware failures.

2. One thing that distinguishes "industrial strength" DBMS's from less capable DBMS's is the extent to which they truly support transactions. (That’s one reason why we’re using db2 in this course.)

3. Relationship to syllabus - the second half of November is given over to topics related to the notion of a transaction.

III. Some important terms we will be using in discussing DBMS's

A. Terms covered so far:

1. Database: all the operational data of an organization (but not transient data such as input and output streams).

2. Meta-data: Data about data, stored in the data dictionary and used by the DBMS to translate higher-level requests into appropriate physical operations.

3. Query language: A facility provided by many DBMS's to allow end users to get at information in the database without a special-purpose program.

4. Three levels of data description:
   a) Physical: how the data is actually stored in files.
   b) Logical (conceptual): big picture of data objects and relationships between them.
   c) View: a custom picture of some of the data for a particular user, involving subsetting and/or virtual fields computed at a lower level.

5. Important concerns:

   a) Data redundancy (to be avoided in most cases): storage of the same item of information in more than one physical place. (This is to be distinguished from deliberate redundancy for backup or robustness purposes - e.g. various kinds of mirroring strategies.)

   b) Data inconsistency (to be avoided always): disagreement between two copies of the same redundantly-stored information.

   c) Security constraints: rules as to who may access what data in what way, stored in the data dictionary and enforced by the DBMS.

   d) Integrity constraints: rules as to what values can be stored for certain data items, stored in the data dictionary and enforced by the DBMS.

   e) Atomicity constraints: the DBMS guarantees that a transaction is atomic in the face of system crashes - i.e. either the entire transaction is performed, or none of it is performed.

   f) Concurrency constraints: the DBMS guarantees that simultaneous access to the same data item by more than one user does not result in the generation of an incorrect result.

   g) Data independence: changes at a lower level of the three levels of data description should not be visible at a higher level.

      (1) Physical data independence: changes at the physical level should not affect the logical or view levels.

      (2) Logical data independence: changes at the logical level should not affect the view level.

B. Instances and Schemas

1. A database schema is a design for a database, showing the kinds of objects in it and the kinds of relationships between them.

2. An instance of a database is a particular set of objects and relationships.

3. For a given database, the schema will not often change (though it will need periodic updates as needs change.) But each change to the stored data affects the instance.

Example: a type in a programming language is like a schema, and the value of a variable of that type is like an instance.

C. Database administrator (DBA): in a database environment, one individual (or on large systems a team of individuals) is given responsibility for maintaining the database schema, defining views etc. This person (or team) is the only one who looks at data at the physical and logical levels; all others use views. It is a very important and senior position. (Note: with a personal computer DBMS the user is his/her own DBA.)

D. Database languages

1. Any DBMS must provide some language whereby a schema can be described and an instance can be manipulated.

2. A data definition language (DDL) is a DBMS-provided language for describing a database schema. It is used primarily if not exclusively by the database administrator.

3. A data manipulation language (DML) is a DBMS-provided language for accessing the data in the database. It is used by application programs and/or end users (in which case it may be called a query language).

   The predominant language used as both a DDL and a DML with relational databases is Structured Query Language (SQL) - which is why this language plays a prominent role in the course. (In fact, databases based on other models are often equipped to respond to queries in SQL.)

   Cf handout of SQL DDL given out earlier for the sample database, plus SQL DML used in the various examples to query and update the database

Appendix - SQL for demonstration database creation

```
create database intro;
create table accounts(
          owner char(20),
          address char(30),
          checking_balance decimal
               constraint positive_checking
                    check (checking_balance > 0),
          savings_balance decimal
               constraint positive_savings
                    check (savings_balance > 0));
create view account_owners as
     select owner, address
          from accounts;
grant select on account_owners to public;
grant update(address) on account_owners to public;
insert into accounts values(
          'Aardvark',
          'Jenks Sub-basement',
          100,
          1000);
```