CPS221 Lecture - Introduction to Database Systems

Last revised October 8, 2012

Objectives:

1. To understand the difference between "data" and "information"
2. To be familiar with key issues such as privacy, integrity, security, and preservation of information.
3. To introduce the notion of persistence and various ways of achieving it
4. To introduce key DBMS concepts
5. To understand that a DBMS can help address issues like privacy, etc.

Materials:

1. Example from Kroenke/Hatch pp 15-18 to read; projectables of figures 1-5, 1-6, 1-7 and 1-8
2. db2 version of database used for SQL use lab to demo
3. db2 security database

I. Preliminary Notions

A. The terms “data” and “information” are familiar, but actually mean two different things. The difference is illustrated by the following example

1. READ example from Kroenke/Hatch pp. 15-18

   PROJECT Figures 1-5, 1-6, 1-7, 1-8 while reading.

2. In this unit of the course, we will be talking about database management systems - that is, systems for recording and providing access to data. But it is important that we set this discussion in the context of the purpose of recording data.

   a) One purpose is, of course, to facilitate some process.

(1) For example, a database at Gordon keeps track of current enrollments in courses and academic history to facilitate the day to day operations of the college and to furnish transcript information alumni will need. (You would be rather unhappy if you applied to a job or graduate school and the college couldn't furnish any record of the courses you completed here!)

(2) Again, a database at an ecommerce site like Amazon keeps track of orders to facilitate shipping what you have ordered and billing your credit card - as well as making recommendations of items you might like the next time you visit the site.

b) But the raw data can also be processed in various ways to provide information needed for operations.

(1) For example, data in the registration database can be used to generate information as to what individuals are close to graduation based on total credits earned, or what individuals have exhibited stellar academic performance (based on a gpa computed from academic history information) and thus merit graduation with honors, etc.

(2) An ecommerce site may use information about what items are often purchased together to facilitate recommendations along the lines of "you might also like ..."

c) Moreover, often the raw data will be processed in various ways to provide information needed for decision-making.

(1) Course registration data can be processed to provide information about under-enrolled courses or courses needing additional sections.

(2) An ecommerce site may also use information about what items are not selling well to facilitate decisions about what items to feature or stop carrying.

B. When dealing with data about individuals, a number of very important issues arise.

1. Issues pertaining to data privacy.

a) In essence, privacy is the right of the individual to control who may access information about them. In the US (and indeed in many parts of the world) privacy laws regulate access to personally identifiable information including medical, educational, and financial records.

b) Of course, privacy cannot be absolute.

(1) Our tax system, for example, requires the disclosure of certain financial information to agencies like the IRS. However, there are regulations governing disclosure of such information.

(2) In the US, certain information about us is considered to be part of the public record, and hence available to anyone. Some examples of things in the public record include:

(a) Vital statistics such as birth and death information.

(b) Property ownership information

(c) Most records of court proceedings

(d) Information about elected officials and government employees, including official correspondence, salary information, etc.

2. Issues pertaining to data integrity.

   a) Integrity is concerned with the accuracy of data.

      For example, with regard to your grades, privacy is concerned with who may have access to them, while integrity is concerned with being sure they are accurately recorded

   b) Data integrity begins when data is first stored in a system. But it goes beyond that, since data is subject to corruption

   c) A classic example concerns the corruption of data that can result from concurrency if appropriate measures are not taken.

      Example:

      Suppose a husband and wife share a checking account. Suppose that, at precisely the same time, one partner is depositing $200 to their checking account, while the other is withdrawing $100. Suppose, further, that the initial balance in the checking account is $1000 - so that the correct balance, after both operations, should be $1100.

      If the software that accesses the account does not adequately deal with concurrency, either of the following scenarios is possible.

      (1) Deposit transaction reads current balance $1000
          Deposit transaction adds $200 to yield $1200
          Withdrawal transaction reads current balance $1000
          Withdrawal transaction subtracts $1000 yielding $900
          Deposit transaction writes updated balance $1200
          Withdrawal transaction writes updated balance $900

          Final balance is $200 too low

(2) Deposit transaction reads current balance $1000
Deposit transaction adds $200 to yield $1200
Withdrawal transaction reads current balance $1000
Withdrawal transaction subtracts $1000 yielding $900
Withdrawal transaction writes updated balance $900
Deposit transaction writes updated balance $1200

Final balance is $100 to high

(3) Issues pertaining to data security.
Security is concerned with protecting data against

(a) Access by unauthorized individuals

(b) Modification by unauthorized individuals.
Clearly, data privacy is not possible if unauthorized individuals can access data which should be private; and data integrity is not possible if unauthorized individuals can change information.

(4) Issues pertaining to data preservation.

(a) There are many situations in which the loss of data can have dire consequences. Consider, for example, what would happen if the college lost the records of the courses you have taken, or your bank lost the records of your bank account.

(b) Many things can result in loss of data: fire, flood, explosion, theft, or even simple media failure.

(c) Of course, data preservation is not only concerned with protecting data against total loss, but also against corruption once it has been stored due to things like partial failure of media or system problems happening during update.

C. The Notion of Data Persistence

1. Thus far, almost everything we have done has involved objects that reside in main memory (RAM) on some computer. This means, of course that those objects "live" only while the program is running, and cease to exist when the program is terminated, either via normal exit or as a result of a system crash, power failure, etc.

   a) This is a consequence of the fact that the CPU can only directly manipulate information that is stored in main memory. Information stored elsewhere (e.g. on disk) must be brought into main memory before it can be manipulated.

   b) Note that access times for current main memory technologies is on the order of 60-70 ns. Access time for data on disk is on the order of 10 ms. Since 1 ms is 1 million ns, this is over a 100,000 to 1 ratio!

2. Obviously, for many applications this is not sufficient. We need some way to make certain objects PERSISTENT - to preserve them between runs of the program.

   EXAMPLE:

   In the registration database example used in a CPS122 lab and in the RMI lab in this course - and which we will use again in a future lab - there is no persistence mechanism - all courses start out empty when we first run the program, and enroll/drop/grade operations are lost when the program exits. Though we've used the program to illustrate many interesting concepts, as it stands right now it's actually useless!

   EXAMPLE:

   Recall the Video Store project you did in CPS122. Which objects need to be persistent?

   ASK

3. Because this is so important, it turns out there are two broad ways of meeting this need.

   a) The approach taken by many familiar applications, utilizing a File menu with New, Open, and Save options

      However, this approach has very serious limitations.

      ASK

      (a) Data is saved only at certain times.

         i) When the user explicitly uses the Save menu option.

         ii) In some cases, automatically via some sort of auto-save facility.

         In either case, if the program crashes or the power is lost, all work done since the last Save is lost. This may be acceptable for applications like a word processor (especially one with auto-save where the information loss in the case of a crash may be relatively small), but is obviously not acceptable for recording transactions in a bank or an e-commerce system.

      (b) It the stored database is large, then an "Open" or "Save" operation can take a great deal of time.

   b) Another approach is to make use of information that resides primarily on disk, with a portion of the information temporarily copied to main memory for access/update, and with changes made to the in-memory copy immediately written back to disk, on a transaction-by-transaction basis.

II. **Fundamental Concepts of Database Management Systems**

A. There are actually two broad approaches that can be taken to providing persistence by storing information on disk: a file-processing approach and a database management system approach. (The latter, of course, is the subject of this unit of the course.)

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. The file-processing approach is characterized by a close relationship between programs and data.

   a) 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: A program written in a C-like language that accesses a file of students might contain or include a declaration like this

```
struct Student
{
     char[7] id;
     char[15] last_name;
     char[15] first_name;
     char[4] major;
};
```

b) As a corollary, any change in the structure of the file will necessitate a change in the program. In particular, if several programs access the same file, then if the requirements for one change calling for adding a field, then all the programs need to change.

Example: Suppose one of the programs needed this file to also contain the student's birth date. Then a field would need to be added to the declaration for `struct Student`, affecting all of the programs (at least to the extent of calling for a recompilation).

c) 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.

d) 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 redunantly - i.e. in more than one place in the database. This, however, creates new problems:

ASK CLASS

(1) 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.)

(2) 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.

(3) 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.

(a) 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.

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

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

ii) The current term.

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

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

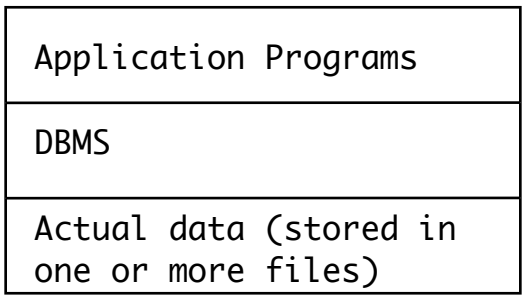
i) In one case, a student changed into the computer science major in mid-term, and I was assigned as her advisor. This was duly recorded in the current term's file; however, the file for the new term had already been created, containing her old advisor's name, and this was not changed.

ii) 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.

iii) 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)!

(4) 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.

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



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.

B. In addition to the data itself, the database maintained by the DBMS also contains META-DATA: data about the data.

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.

   student.id, student.last_name, student.first_name, student.major

   which specifies where the data item is stored (what table it is in, and what column in the table), so that the DBMS can locate it.

2. DEMO:

   Open a terminal window to system db2 version of registration database; widen it

```
db2 -t
connect to cps221 user bjork;
list tables;
describe table student;
```

C. A DBMS can facilitate addressing the key issues of privacy, security, integrity and preservation that we looked at earlier.

1. The above demonstration showed only part of the meta-data for student. In addition to the data type information, the meta-data may include integrity constraints

   Often, the values of certain items in a database are logically constrained to only certain possibilities.

   (1) Example: in the student table, the field id is declared to be the primary key - which implies that no two rows can contain the same value - and also declared to prohibit a null value

Demo:

```
select * from student;
insert into student values('7777777',
    'Gopher', 'Gertrude', 'ART');
insert into student values('1111111', 'Horse',
    'Horace', 'CPSC');
```

(2) Example: a grade field in a registration system may only contain values like A, A-, B+ ... D-, F, 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.

DEMO:
```
insert into course_taken
    values('1111111', 'BCM', '103',
    '2009FA', 4, 'Z');
```

(3) Good use of integrity constraints facilitates preserving data integrity - one of the key issues we looked at earlier.

2. Another type of information that may be present in the metadata is 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, a student might be allowed to see (but not change) only the grades he/she has earned; the registrar might be allowed to both see and change the grades of any student

DEMO:
```
connect to security user bjork;
set schema registrar;
select * from course_taken;
update course_taken set grade = 'C+'
    where id =  '5555555' and
        department = 'BCM' and
        course_number = '101';
select * from course_taken;

connect to security user aardvark;
set schema registrar
select * from course_taken;
select * from student_info;
update course_taken set grade = 'A'
    where id = '1111111';
```

c) Good use of security constraints can facilitate both preserving data privacy and data security - tw more of the key issues we looked at earlier.

3. A DBMS can take care of concurrency issues without the various programs accessing the same database even needing to be aware of each other - we discuss this in CPS352 .

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 data automatically.

DEMO:

start two "bjork" connections to accounts using db2 `-t +c`
widen windows
in both windows: `select * from accounts`

Now 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 number = 42;
update accounts
     set savings_balance = savings_balance - 100
     where number = 42;
```

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.

4. Finally, a DBMS can help take care of issues pertaining to system crashes, backups, etc in such a way as to ensure there is no loss of data for transactions that have already completed. Again, we discuss this in CPS352.

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

1. 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 enrollement in all courses in our sample registration database, 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:

```
count = 0.0;
for each record in the enrollment file
   count ++;
```

   - Code to print the final value of count

   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!)

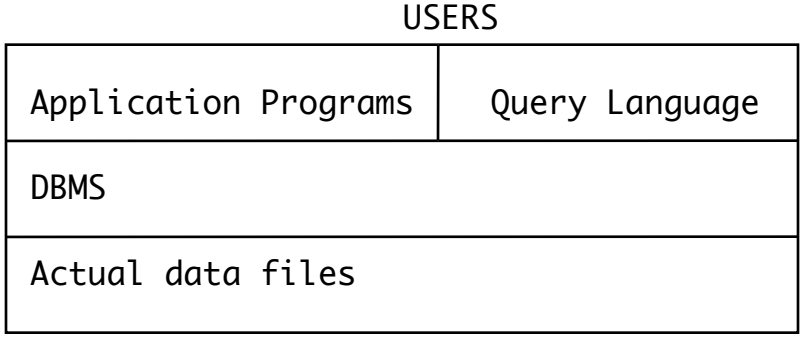
2. Most DBMS's also include a QUERY LANGUAGE which allows a moderately sophisticated user to get at information in the data base directly, without going through an application program.

   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:

```
connect to cps221;
select 'Total enrollment is ', count(*)
           from enrolled_in;
```

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

b) 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.)

3. Recall the distinction we made earlier between "data" and "information". The query interface allows users of the database to not only access the raw data, but also to perform various operations that convert it into useful information.

E. 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.

1. 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 in CPS352).

2. In assessing the performance of a DBMS (or any system that services multiple users), some of the factors we looked at in conjunction with Operating System scheduling turn out to be relevant again.

a) The notion of throughput.

What do we mean by this?

ASK

In the case of a DBMS, we need to ensure that system throughput is adequate for the demand. Consider what would happen to an ecommerce site, say, if the throughput of its database system were less than the rate of customer transactions!

b) The notion of response time

What do we mean by this?

ASK

Again, with interactive users, this can be critical. Who would choose to make use of an ecommerce site if the response time to queries were consistently too slow?

F. Finally, we should note that, in this unit of the course, we will be focussing on what you might call "traditional" DBMSs that deal with information that takes the form of numbers and short character strings.

However, we should note that there are specialized databases that deal with other sorts of information - e.g.

1. Multimedia databases that store video or audio files.

   Example: YouTube

2. Document databases that store large text files.

   Example: the various journal databases maintained by the library

3. Information retrieval databases

   Example: Google