CS 325 - DB Reading Packet 1: "Database Processing and Development"

SOURCES:

- Kroenke, "Database Processing: Fundamentals, Design, and Implementation", 7th edition, Chapter 1, Prentice Hall, 1999.
- Connolly and Begg, "Database Systems: A Practical Approach to Design Implementation and Management", 3rd Edition, Addison-Wesley.
- Rob and Coronel, "Database Systems: Design, Implementation, and Management", 3rd Edition, Thomson, 1997.
- Sharon Tuttle, “Database Design”, Lecture readings, Cal Poly Humboldt

Intro to Databases

Let's start with the question: **What is a database?** We'll work our way to a more technical definition, but to start:

a (relational) **database** is a *collection* of relations, or tables, holding information about different *interrelated entities*.

In a later packet, we'll discuss this idea of an **entity** -- some collection of instances of something important in a world or scenario -- and we'll eventually see that an entity can be represented by one or more relational tables in a relational database.

But before we go on, let's make sure something is clear: is Oracle a database? is Excel? is Access?

Answer: **No**. Excel is a spreadsheet program, and so doesn't belong in our discussion for this class at all. But you'd be surprised how many people think it is a "database program", because it often looks so "tabular" -- and you'd also be amazed at the lengths people go to do things in Excel that would be more easily done with database software.

Why am I claiming that Oracle and Access are not "databases", though? Because, really, they are one level up from that -- in the world of database design, programs/products such as Oracle, Access, MySQL, PostgreSQL, SQL Server, Informix, and many more are **database management systems**, often abbreviated as **DBMS**s. A DBMS is a program or collection of programs that acts as the *interface* between a user or application program and a database. (Although, that said, we may describe a database managed by an Oracle DBMS as an Oracle database, a database managed by an Access DBMS as an Access database, and so on.)

My favorite parallel is to think about operating systems, and how one can consider the operating system as software that serves as the interface between the user or an application program and a computer's hardware. Likewise, a DBMS serves as the interface between the user or an application program and a database.

This DBMS, this interface, manages the database structure, controls access to the data stored in the database, and provides various tools to allow users/application programs to use the data. It provides a useful illusion of how the data is organized to people and to application programs, hiding the actual physical details of how the data is actually stored.

Before going on, let's consider the term *database application*. This is simply an application program that makes use of a database. CS 328 has as its focus the programming of such database applications.

Note that a database may be used at a variety of levels -- for example:

- personal - one concurrent user
- workgroup - say, less than 25 concurrent users
- organizational - say, 100s of concurrent users

The Relationship between Application Programs and the DBMS

You'll recall that we just mentioned that a user or an application program may interact with a DBMS. It is certainly possible, when an application program interacts with a DBMS, that a person may be interacting with that application program in turn (that is, a person may interact with an application program that interacts with the DBMS to actually access database data).

Some DBMSs provide such a variety of tools for users that the line between the DBMS and application programs can become increasingly blurred -- consider, for example, a DBMS that offers features such as built-in forms tools, graphical query-by-example, and so on. Although we are talking mostly about database design and implementation this semester, you should remember that many users will not interact directly with the database tables via the DBMS, but will instead interact via various forms, reports, and queries within applications that interact with the DBMS.

Another point, before we proceed. Note that you are often not given the source of a DBMS (at least, not for a commercial DBMS); you generally cannot change or modify the DBMS itself. By "database design", then, we mean "the design of the database structure that will be used to store and manage data rather than the design of the DBMS software".

But, if you have a sufficiently powerful DBMS, why are we so concerned about database design? Because, just as a wonderful, rich operating system does not guarantee that a programmer will not write a scummy application program that runs under that OS, a wonderful DBMS does not guarantee that a poorly-designed database will run well when managed by that DBMS. "Even a good DBMS will perform poorly with a badly designed database."

History Part 1: from File-Processing Systems to Database Systems

File-Processing Systems

(Kroenke, p. 10) It is useful "...to look at the characteristics of systems that predated the use of database technology….", to "reveal the problems that database technology has solved", or at least alleviated.

The first business information systems stored groups of records in separate files, and were called file-processing systems. It is useful at this point to point out that the concept of a **file** has had more than one meaning in computing -- not all files are the "streams" of characters you are probably used to from the popular stream-based file input/output packages available in such languages as C++ and Java.

An alternative view, more common in languages such as COBOL, is of a file as a collection of related **records**, where each record is a collection of logically-connected **fields**, and where each field is a character or a group of characters. This view, of a file as a collection of related records, is the more traditional "mainframe" view, I'd say; and it is also the area from which databases "grew".

So, imagine it is the 1960's, and companies are (p. 16, Kroenke) "producing data at phenomenal rates in file processing systems" … and, they were finding, as the sheer quantity of data increased, that "the data were becoming difficult to manage, and new systems were becoming increasingly difficult to develop".

In short, they were bumping right into some important limitations of file-processing systems -- limitations that become apparent as the amount of data involved increases:

- separated and isolated data
- (unnecessary) data duplication
- application program dependence
- incompatible files
- difficulty of representing data in the users' perspectives

And note how the above list is not complete, by any means -- consider, for example, dealing with implementing security or strategically sharing data when it is all stored in such record-based files.

Consider the following example, as we discuss the above limitations. Consider: you have a collection of restaurants, some that you already like, some that you are interesting in trying. Let's say that you categorize these in different ways -- for example,

- by price
- by type of food
- by location
- by hours (open for breakfast? at 3:00 am? etc.)
- by methods of payment accepted (cash only? credit cards OK? checks OK?)

...to make it easier for those times when you want to go out to eat, but are having trouble coming up with where you want to eat.

One way to handle this might be to store this information in different files. You might have, say, a list of Italian restaurants in one file, restaurants that are open all night in another, restaurants that take credit cards in a third, pizza places in a fourth, restaurants with entrees less than $5 in a fifth, and so on. Keep this example in mind -- we'll be coming back to it as we discuss the file-processing shortcomings listed earlier.

Separated and isolated data

Data that is separated into different files -- isolated from the data in other files -- can make that data difficult to relate to each other when that might be desired. In our small example, how would you write a program to give you Italian restaurants that are open all night, from these files? It's not impossible, but it is difficult, and likely just enough bother to discourage maximum use of one's data.

(Unnecessary) Data duplication

Consider our restaurant files. The same data may be stored in numerous different files, different numbers of times: for example, the address and phone number for an inexpensive pizza place that is open all night might appear in several different files. As another example, consider a rental client at a DVD rental store -- if such a company stored its information in a file-processing system, imagine how often a customer's address, phone number, or contact information might be stored: once for a rental file, maybe again for special orders, maybe again for special promotions, and the list goes on.

And, the extra space this takes is not as big a problem as the potential for data integrity problems that can occur as a result of such unnecessarily duplicated data. Consider: what if a restaurant changes its phone number? What if a customer moves? What if you do not change all of the copies of such information? And what if a restaurant closes? I should, then, remove all mentions of it.

Kroenke discusses how the idea of data **integrity** is not so much that the data in a collection, whether of files or within a database, is correct, but that it is logically **consistent** within that collection of data. Can you see that if one file lists the phone number of Larry's House o' Pastry as 555-2827, and another lists its phone number as 666-2827, then the data in those two files is not logically consistent? And -- which, in that case, is more likely to be the correct phone number?

Application program dependence

In a file-processing system, application programs tend to heavily depend on the file formats -- that is, the physical formats of files and records become an inherent part of the application code.

Why is that such a problem? Consider: what if the file format changes? Say, an area goes from 7-digit telephone dialing to 10-digit dialing, or you decide to store zip codes with zip-plus-4 instead of plain 5-digit zip codes. Such changes often require changing every application program that uses a file whose record format has been changed, maybe even if it never uses phone numbers or zip codes. When you change the physical format of a file, you may have to change all applications that use that file, even if the change is to a field not used by that application.

Incompatible files

This is a little harder to understand now that we are so used to stream-based ASCII or Unicode files, but record-based files' format often depended on the language or product used to generate them. The result was that files from different "sources" might end up unable to be readily combined or compared, or difficult to process jointly.

Difficulty of representing data in the users' perspectives

This issue is simply noting that, in a record-based files system, it is difficult to represent the data in a form that seems "natural" to users, not to mention quite hard to do so quickly, or on a whim (to follow a hunch, for example). It is hard to support **queries**, particularly so-called ad hoc queries. (What is a query? In the simplest sense, it is simply a question; and an ad hoc query, then, is a "spur-of-the-moment" question, one you want to ask after the data is there, that you didn't necessarily have in mind when gathering the data or organizing it into files.) This discourages creative use of one's data -- and makes it very hard for others to use such data, as they have to know how someone else organized it into files, what format they used, and so on.

We are not saying, here, that storing data in files is simply and irredeemably awful; obviously, we still use files happily and frequently. What we are saying is that, as the quantity of data grows, it gets harder and harder to deal with data that is stored in files. That is, we would like an alternative to files for collections of data that may be used in different ways by different people and different applications, especially when it may be used on a day-to-day basis within some setting, and especially if one wishes to encourage brainstorming and other creative uses of such data.

Database Processing Systems

So, database technology was developed largely in answer to these limitations of file-processing systems. Basically, a major "big idea" here is that lower-level, "file" level details should be hidden from application programs by the DBMS. That is, the application program deals with the DBMS, not the actual files containing the data, and so the application program does not have to know or care about the actual physical format in which the data is stored.

Hopefully, then, a database processing system has at least the potential to have the following advantages over the file-processing approach:

- integrated data

- reduced (unnecessary) data duplication
- decreased application program dependence
- easier representation of the users' perspectives

And, there is the potential for other benefits, too, including (but certainly not limited to) improved security, more sharing of data, economy of scale, and more.

Integrated data

In a database system, all of the application data is stored in a single facility called the database, managed and accessed via a DBMS. A programmer does not need to write programs to consolidate the files, or to try to relate data that may be related in different files -- instead, the application programmer indicates what is needed, and the DBMS takes care of the actions required to do so. Instead of a collection of files, which may or may not be related, in a database-processing system all of the data is conceptually in one "place", the database.

Reduced (unnecessary) data duplication

It is important to understand that data duplication is **not** eliminated in a database. But, in a **well-designed** database, **unnecessary** data duplication can be reduced, and in such a way as to potentially increase data integrity. In a well-designed database, you generally store most information just once, but because the data is related and integrated it can be processed and related in various ways via the DBMS as needed. And, it is reduced without losing the possibility of finding the phone number of a restaurant, or the address of a customer, etc., regardless of the many contexts that a restaurant or customer may be involved. When data duplication is reduced, that likewise reduces the chances of data integrity problems due to changing one copy of a datum but not all others.

Note that we're going to find, during the course of the semester, that sometimes we use data duplication deliberately for beneficial ends -- that duplicated data is not always bad -- but, we do want to avoid unnecessarily duplicated data, and we'll be discussing what that means in some detail.

Decreased application program dependence

Because a DBMS essentially hides the actual physical file format of the data from users and application programs, database processing reduces the dependence of programs on those file formats. A DBMS could completely change how the physical data files implementing a particular database were formatted, and application programs using that database would not be affected as all as long as how those applications request data from the DBMS does not change. The DBMS really insulates the application programs from caring about the actual, physical format of the data. Application programs are dependent on what they get from the DBMS, not on however the data within the underlying database is stored. Better still, with many DBMS's, the application program can specify, to at least some degree, what they want, and in what format!

(Really, this is an example of the information hiding concept that you hopefully heard about when introduced to object-oriented programming and abstract data types!)

The benefits here can go even further -- for example, consider something like Java's Java Data Base Connectivity (JDBC) Application Programming Interface (API), which allows you to connect to literally dozens of different DBMSs with a single mostly-common interface.

Easier representation of the users' perspectives

In a relational DBMS -- which will be our primary focus in this course -- thanks to standardized query languages and powerful abstractions such as relations, such a DBMS can more easily represent the

objects found in the user's world, in the user's perspective. And, many DBMSs' provide tools to get different views of related data, from different viewpoints; we'll discuss some of these possibilities further as the semester progresses.

But database processing isn't all advantages...

This is not to say that database processing brings nothing but advantages, however. Most complex things have both advantages and disadvantages.

For example, DBMSs can be very complex, and can get very large from a memory point of view (as can the database itself). Note that one **cannot** say definitively, for a given collection of data, whether storing that data in files or in a database managed by a DBMS would take less room; one can describe scenarios that could go either way.

DBMSs can be very expensive, and it can be likewise expensive to convert data into a database.

One also cannot state definitively whether the performance would be better for an application dealing with a DBMS than for one dealing with files; a DBMS is designed to be more general, trying to satisfy the needs of many, whereas a custom application can be highly optimized. Also, when large quantities of data are being streamed constantly into storage, sometimes the DBMS overhead is too time-consuming to keep up.

And of course centralized data introduces the potential for a central point of failure -- backups and security for centralized data have to be taken seriously.