



Introduction to Databases

Orlando Karam
okaram@spsu.edu



SOUTHERN
POLYTECHNIC
STATE UNIVERSITY

Since we will be talking about databases, it is a good idea to define what a database is.

Simply, a database is a collection of data; however, you don't usually see collections of data just lying around on the streets ; we normally use a database application to access that data; a database application is a computer program designed to provide access to a specific database; when you go, say, to Amazon.com, you see a web

site, that allows you to buy books; that's the application; behind that application there's a database, a bunch of data about the books, and about who has purchased them.

Early on, we've realized that, although each application is different, db aps share a lot of functionality, so, as we usually do in computing, we've abstracted the data-access functionality into a particular piece of software, which we call a DBMS, a Database Management System. Examples of DBMSs include Oracle Database, IBM's DB2, PostgreSQL, MySql and MS SQL Server.

Notice that normal people use the term 'database' to refer to database applications, and sometimes to DBMSs too; but for this class, it is useful to keep these three concepts separate.

Basic Definitions

SPSU

- Database (DB) - collection of data
- DBMS - generic program to manage databases
- DB Application - App with specific purpose that accesses one database

We said that a database is a collection of data; so now we need to define data, and other useful concepts derived from data.

Data is each of the basic, raw facts that we enter into a database; however, when designing an application we want to get more than what we put in; we want to be able to get information; that is, data that has been processed to make it more useful; for example, if we're doing a

student management application, the fact that you get an A in this class, is data and would need to be entered; the fact that your GPA is 3.9, is information and does not need to be entered; it can be generated by the application itself by looking at all the data about your grades.

When we're analyzing the data requirements for a particular application, we do not usually focus on specific pieces of data; we want to define classes of data with common characteristics; that is, we do not yet want to know whether you get an A in this class, but the fact that we will be recording facts of the kind: *A student gets some Grade in a Class*; that is, when we design applications, we will be dealing with *metadata*, data about other data; eventually, when people are actually using the system, they'll worry about the actual data; when we're designing it, we worry about the *metadata*.

When talking about metadata, it is useful to distinguish between computer metadata; metadata that can be directly used by a computer, like the data type of a field, or its size, and *human* metadata, metadata that is useful to people, but not directly usable by computers, like the origin of the data, or who decides which value goes into the field.

Another common way to refer to metadata, is to talk about the *schema* of a database, or its *intension*, to refer to all the possible kinds of data that can go into the database; basically its metadata; and to use the terms *database instance* or the *extension* of a database, to refer to the actual data it contains at a given moment in time.

Data etc

SPSU

- Data - basic facts
- Information - processed data
- Metadata - data about data
 - Computer and Human metadata
- Schema (intension) - description of data (metadata)
- Instance (extension) - actual data

Have you ever written an application that needs to directly read and write from a file ? You probable have do you remember it ? ... Was it hard ? ... Was it Painful ?

Chances are you answered yes to all this questions; accessing files directly is hard and tedious, because your programs need to know the format of the file; you need to worry about reading two bytes from the file, and mapping it to an integer in your program, and make sure you consume the whole record before going on to the next one ... You can do it, but it is a pain ...

Now imagine what happens if you change the file format; you now need to change all the programs that access that file; and, what happens to your old files ? you need to write a translator program to transform the old files to the new format; so, what does this mean ? you don't change the file format unless you really really need to, right ?

We used to write data applications that way, and there are still some of those running in mainframes; but we realized it made sense to keep the metadata with the data, and to have an abstraction layer that reads the metadata and knows how to deal with the data, thus providing program-data independence; over time, this layer evolved into the modern DBMS.

Nowadays, most DBMSs run as a server process, with the applications communicating with this server over the network; this allows for one DBMS to support many applications, and for different applications to access the exact same data. We also use a standard database language, SQL, so even if you change your DBMS vendor your application won't need to change, or at least not much.

DB vs File System Apps

SPSU

- Apps that deal directly with files have **program-data dependence**
 - the program needs to know the format of the data
 - if the format changes, the program changes too !
- DBMS provides **program-data independence**
 - DBMS knows data format, programs do not !
- DB Apps usually include
 - SQL - Standard query language**
 - Client-Server - remote access, centralization**

The main technical advantage of writing database applications (as opposed to file-processing ones), is program-data independence; nowadays, we also do client-server applications, which together with independence, allows for much greater centralization.

Program-data independence usually entails much less maintenance, since we do not need to change the programs, even if the format of the data changes, either for performance reasons, or by adding more fields.

Now, together with the right organizational policies, centralization enables us to have the data only once, minimizing redundancy; it also allows us to focus resources to ensure data consistency and greater data quality.

If the data is in one place, and it can be accessed over the network, then sharing it is much easier; we may even let the users (or their resident computer geek) execute SQL queries directly against the database.

Notice that other than most advantages are not technological, they need the right policies within an organization and so are subject to politics, bad management or simply a competing set of requirements that do not allow this to happen.

DB Apps Advantages

SPSU

- Advantages

- Program-data independence

- which entails Reduced program maintenance

- And enables data centralization, and so

- Less data redundancy
 - Improved data consistency, quality, standards
 - Improved Data sharing
 - Improved data accessibility and ad-hoc queries

Now, Databases also have some costs over file processing applications; first of all, we now need to obtain a DBMS, but that's easy nowadays, with many high-quality DBMSs, like PostgreSQL or MySQL available for free; however, we need to maintain and administer this software, which entails the need for a database administrator, or DBA; most system administrators are able to do this job with not much trouble, but it is still one more thing to do, one more software to secure etc.

The DBMS also needs specialized backups, as opposed to just copying files; but nowadays most DBMSs provide a simple way to backup their data; however, it is, again, one more thing to worry about.

The benefits of the DB approach greatly outweigh the disadvantages for most organizations, which is why it is the default way of developing software nowadays.

DB Apps Disadvantages

SPSU

- Special software, DBMS
 - But simple nowadays
- Entails DBA
 - But most sysadmins can do it, not perfectly but ...
- Specialized backups etc
 - But again, no big deal usually

It is sometimes useful to classify database applications by their size, especially since they oftentimes use different technologies.

The smallest applications would be called personal, and they're designed for one user and provide no network connectivity. We tend to not do many of these anymore, since we've learned to do client-server applications almost as easily, and the advantages of connectivity outweigh the costs.

Even your PDA now gets data over the network.

Probably the most prevalent are workgroup applications, which are designed to be used by a relatively small group of users, less than 50 or so, with most of them on the same building, and most connectivity over a local area network.

If you add more users and more sites, you get enterprise applications; these usually affect all users in a big organization and they are often mission critical; if they go down, the company cannot function, so we put a lot of effort to keep them running.

Nowadays, many applications are designed for the internet; these have many more users, and of course the connectivity is through the internet; many times the users will be casual and the organization has no control over their computers, so we tend to use cross-platform technologies like HTML.

We've now learned to do web applications easily, so many times they are the default model; even if they're going to be used only by 20 users in the same workgroup, we will build the application as an internet application, since it is easier, and means less hassle for the user.

DB Application size

SPSU

- Personal DB Apps - one person, no network
- Workgroup DB Apps - small group, LAN
- Enterprise DB Apps - mission critical, many users, WAN
- Internet DB Apps - many users, probably casual users, over the internet

One important feature of most DBMSs is the client-server programming model; the DBMS is implemented as a server program that accepts SQL commands over the network and sends data from the database in return. This allows for centralized control of the data, and many flexible applications.

The clients can be general clients, used by programmers, which allow access to any kind of database, or can be database applications, built for a specific database.

As far as the DBMS is concerned, there is only one kind of client; all clients send SQL commands and receive data; however, for the end user the experience may be quite different. In a way, many database applications are just a fancy interface on top of SQL commands !

Client-Server

SPSU

- DBMS server, controls access to database
- Clients use network connection to send SQL, receive data
- General clients (GUI or Text) - for programmers or DBAs
- DB Apps are just another client - used by end-users

8

With pure client server applications, the client application is still a full program running on the end user's computer; this implies that it has to be installed on those computers, and that when it is updated, it needs to be updated on all the client computers.

An easier way, is to have a more generic client, providing a generic user interface, and add another layer in between. Of course, nowadays, this generic client is a web browser.

Three-Tier apps

SPSU

- Add another layer between client and server
- Middle tier is business logic
- Client can be very simple (browser)
- As far as DBMS, client is business tier
- Updating business logic tier is easy

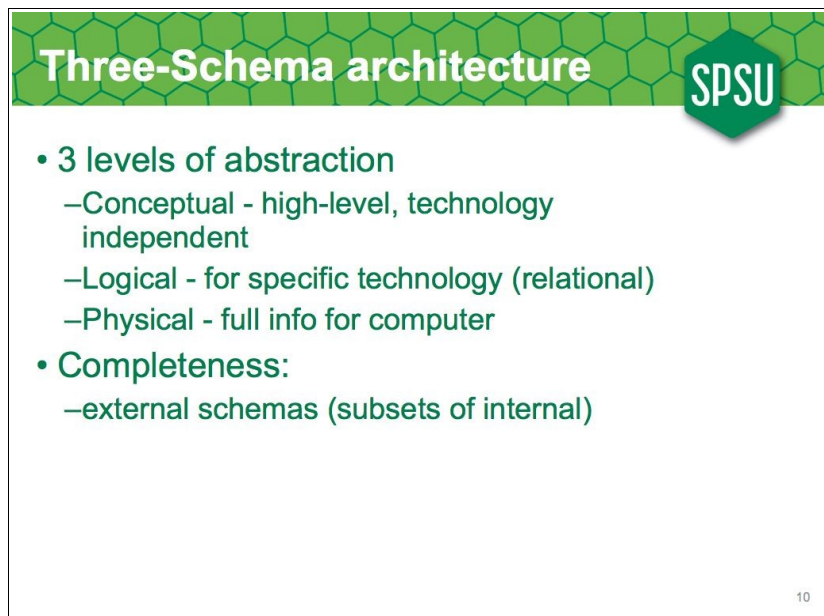
9

So, the browser access the application, which generates HTML; this application runs in a separate server, and it is this application that accesses the database; as far as the DBMS

is concerned, the client is the middle-tier, it never knows about the browser.

When we create database applications, we've found it useful to look at the database schema, that is, the kinds of data that go into the database, at 3 different levels of abstraction.

When we analyze the requirements for the application, we want a highly abstract view that allows us to easily communicate with the users who are providing those requirements; this is the conceptual schema.



The slide features a green hexagonal pattern header with the title 'Three-Schema architecture' in white. The SPSU logo is in the top right corner. The main content is a bulleted list in green text. A small '10' is in the bottom right corner.

Three-Schema architecture

- 3 levels of abstraction
 - Conceptual - high-level, technology independent
 - Logical - for specific technology (relational)
 - Physical - full info for computer
- Completeness:
 - external schemas (subsets of internal)

10

We then map that conceptual schema to a particular database technology (usually relational) so that it can be implemented; this is the level at which programmers communicate and it is usually fairly abstracted; program-data independence means that I can just think of a data type as an integer, and not need to worry much about whether it is a 2-byte or a 4-byte int.

Eventually, the database will go into an actual hard drive; the DBMS needs to know whether that in is two or four bytes; this is the physical schema, and is usually only seen by the database administrators.

As an orthogonal issue, we oftentimes find it useful to provide subsets of the schema for different kinds of users; we call this external schemas or user views.

Although it is not even close to what it really happens, it is useful to have a simplified representation of how we develop software.

In this ideal world, we first do requirements analysis, to find out what the users really want; for db apps, we do conceptual modeling to figure out what data needs to be in the database.

After we have the requirements, we design how we're going to implement them; for databases applications, we would transform the conceptual schema into a logical one, and eventually into a physical schema.

Finally, we implement both the application and the database, and eventually we may need to do maintenance.

