

An Overview

A database is based on *data*, which carries information, the lifeblood of every enterprise.

There are essentially two aspects of data: a logical one and a physical one. The logical aspect provides a *meaning* of a piece of data, and a physical addresses the *storage* related issues.

For example, the number 108 could mean the temperature in Phoenix on some day during this summer, the product of 1^2 , 2^2 and 3^2 , the number of a local road, a room number, etc.. On the other hand, it takes a byte to store this number.

We once put all the data in paper, now they are kept in various kinds of storage media, in terms of bits, bytes, fields, records, and files. A database is a collection of *related* files. A database management system (DBMS) manages such related files.

Why do we need data?

Every enterprise, such as a company, a college, etc., keeps lots of data for various purposes. A college keeps students' transcripts to maintain academic records; enrollment data, i.e., how many students applied, get admitted, paid deposits, enrolled, at what time, etc., to maintain a business record, and, of course, other stuff, such as parking violation record.

Because of the huge amount of data we have to work with, the boring nature of the related processing, as well as the mechanical nature, today, almost everybody uses computers to do the data processing, often via database applications.

We are going to learn *some* of the basic concepts and practical skills involved in database applications in this course.

DBMS

DBMS is a highly sophisticated piece of software that is used to manage all database related applications, including all the involved files, mainly data files and *index* files. A key issue here is that all these files are *related*, thus we have to understand this nature of data relation to have an efficient system.

The relationship among data, *data modeling*, was often thought to be *hierarchical*. Examples include the organization chart, the menu structure, etc. Sometimes, things can be a bit more complicated. For example, a manager manages employees, while the manager herself is also an employee. Thus, we can generally use a *network* model.

DBMS based on these two models were once quite popular, such as IMS by IBM to support the Apollo program in the 1960's.

Relational model

Edgar Codd had a different idea. He wrote a few papers in the 1960's and 1970's, and suggested to look at the relation between data in terms of mathematical set theory. Codd won the Turing Award in 1981 for this work.

Oracle is among the companies that picked up Codd's idea and built much more satisfactory DBMS based on this RDB model.

Today, RDB is the dominant data model in this business, which is also what we are going to learn.

There are tons of DBMS in the market. They all try to address such issues as how to set up a database, i.e, the DDL (Data Declaration Language); and how to design *queries* to get the information we need, e.g., Who has the highest, and the lowest, GPA among her peers? This latter part is often called the DML (Data Manipulation Language).

What will we use?

All these RDB based systems provide various features according to the SQL (Structured Query Language) standard which started at 1989 with the most recent one being SQL 2007, although not completed, and SQL 2011 is already on its way. (<http://www.jcc.com/sql.htm>)

From a user's perspective, these systems are more or less the same. The difference lies in the interface, internal power, features, and reliability. All come down to the price.

We will use *MySQL* (<http://www.mysql.org>) in this course since it does its job reasonably well, it is free and (thus) widely used, although it only partially implements the SQL standard in an incremental manner.

The essentials

Since we often put in huge amount of data into a database, we want to have a database that contains all the information we need, but *with as little fat as possible*. The *normalization theory* can guide us through the whole process of cutting down as much fat as we wish with more and more effort.

On the DML side, the *relational algebra* will guide us through the process of writing correct and efficient SQL queries.

The related topics, and their applications, are essentially covered in Chapters 3-8. In particular, Chapters 3, 5 and 6 provide the very foundation of the RDB theory, Chapter 4 a nice tool of setting up the database, and Chapters 7 and 8 provide some examples of application.

MA2200 or MA3200

Both the normalization process and the relation algebra depend heavily on such set related operations as union, intersection, Cartesian product; subsets, relations, and functions.

That is why these two courses are listed as one of the prerequisites of this course.

If you are not that familiar with them, it is a great idea to check out a book on these subjects from the library.

You can also do a google to check them out.

Let's do a (p)review of mathematics in its minimum.

Little (p)review

A *set* is a collection of objects. For example, we have a set of apples, a collection of chairs, a bunch of students, and a set of prerequisites for this course: Java and Finite (Discrete) Math.

Let A, B be two sets, we can apply the following *operations* to sets,

$$\text{Union } A \cup B = \{x | x \in A \vee x \in B.\}$$

$$\text{Intersection } A \cap B = \{x | x \in A \wedge x \in B.\}$$

$$\text{Complement } A - B = \{x | x \in A \wedge x \notin B.\}$$

and a *relation* to sets:

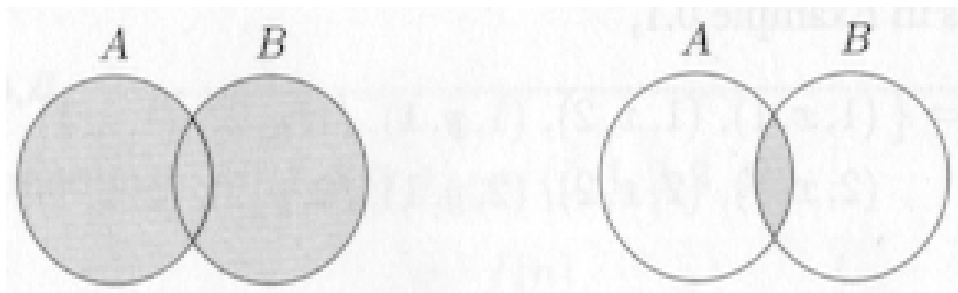
$$\text{Subset } A \subseteq B \equiv \forall x, x \in A \Rightarrow x \in B.$$

Let A be $\{1, 2\}$, B be $\{2, 3\}$. We have that $A \cup B = \{1, 2, 3\} = B \cup A$, $A \cap B = \{2\} = B \cap A$, $A - B = \{1\}$, but $B - A = \{3\}$ and $A \not\subseteq B$.

Visually speaking

Venn diagrams are often used to specify the relationship among sets.

For example, the following depicts the union and intersection operations.



Sequences and tuples

A *sequence* of objects is a list of these objects *in some order*. For example, $(7, 21, 57)$ represents the sequence 7, 21, 57.

Question: What is the difference between a set and a sequence?

Answer: Order.

A finite sequence with k elements is usually called a k -*tuple*. Particularly, a 2-tuple is called a *pair*.

For example, $(7, 21, 57)$ is a 3-tuple, while $(7, 21)$ is a pair.

Cartesian product

This is something that we use quite a bit in RDB to get things mixed up.

Let A, B be two sets, we can define the *Cartesian product* on A and B as follows:

$$A \times B = \{(a, b) | a \in A \wedge b \in B.\}$$

For example, if $A = \{1, 2\}$ and $B = \{x, y, z\}$, then

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}.$$

Question: What is $A \times B \times \{a, b, c\}$?

Question: What is the size of $A \times B \times C$, in general?

Answer: $|A| \times |B| \times |C|$.

Function

A *function* takes in one or more input(s) and sends out an output, e.g., $f(a) = b$. A function is a special *mapping* in the sense that the following always holds: for all $x, y, x = y \Rightarrow f(x) = f(y)$.

Let f be a function, the collection of its possible inputs and outputs are called its *domain*, and *range*, respectively.

$$f : D \rightarrow R.$$

For example, the absolute value function, *abs*, is obviously a function in this sense. We have that:

$$\begin{aligned} \text{abs} : \mathcal{Z} &\rightarrow \mathcal{Z} \\ \text{abs}(x) &= \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{otherwise.} \end{cases} \end{aligned}$$

The above also provides a procedure to compute $\text{abs}(x)$ for any x .

Relation

Now, we come to the central idea of RDB, *relation*.

A *predicate* is a function whose range is $\{T, F\}$.

Since $x \leq y$ is either true or false, depending on if x is no more than y , ' \leq ' is a predicate.

A *predicate* whose domain is a set of k -tuples is called a *relation*. Thus, ' \leq ' is also a relation, since its domain, the collection of all the pairs (x, y) such that $x \leq y$, is a set of 2-tuples.

The gist of RDB

Let R be a n -ary relation, (a_1, \dots, a_n) be a n -tuple, $R(a_1, \dots, a_n)$ means that the latter is true.

If “A freshman John Doe, with id 1111, does live in 123 Main St..” is true, we can represent it as $(1111, \textit{JohnDoe}, 123\textit{MainSt.}, \textit{Freshman})$.

In general, we can have a predicate “A person with Name X , Id Y , status Z , lives in A .” and then collect all the true statements based on this predicate *at the moment* as a relation and represent it as the following *table*:

Id	Name	Address	Status
1111	John Doe	123 Main St.	Freshman
2222	Mary Smith	1 Lake St.	Freshman
1234	Joe Blow	6 Yard Ct.	Junior

This is the mathematical basis of RDB.

The other stuff

Again, since we have to deal with huge amount of related data, we want to organize the related files in such a way that we can get out the needed data, making use of their relationship. This is essentially the topic of Chapter 9.

Chapter 10 is on the entire process of query evaluation, sort of the basic working process of the SQL interpreter.

Chapter 11 is on data recovery, since we bet our last penny on the database.

DBMS is quite expensive, thus it is used by many people during the same time, i.e., concurrently. This may cause some side effects. Chapter 12 studies the related issues.

Finally, RDB, albeit popular, does have some restrictions. Object database, the subject of the last chapter sheds some light from a OOP perspective.

Practically speaking

Once the database is set up, we will try to get out information by *querying a database*, i.e., *applying operations on those relations as sets*. SQL queries only get the data, usually a set of records. We can pose queries, one at a time, in the SQL prompt, but this is not the norm.

To have a sophisticated application, such as an interactive session (WebReg), we also need to have other features, such as the conditional, and the loop structures. We thus often embed SQL queries in a *host language*, which provides the aforementioned general features.

In the past, people used such languages as C++ and Java as the host language, and we will use the WEB friendly *PHP* (www.php.net) because...

... another very important issue is the presentation part of the database application. This part gets in the data from an end user, and throws back the result after certain, hopefully correct, manipulation.

In today's world, there is no better and/or more popular platform than WEB. Thus, many of the database applications directly use the HTML format to write up their presentation platform. We will do the same.

To summarize, practically, we use *HTML* to write up a WEB friendly interface, embedded with database programming code written in *PHP*, which sends queries to, and receives the result from, (a) *MySQL* based database(s).

On the lab work

This course hopefully will provide a solid background in DB related concepts and a strong DB programming component. Thus, besides the textbook, there are two separate, but related, lab notes, for the latter effort.

Considering the amount of the work involved, although there will be a roughly once-a-week lab, students are expected to spend a significant amount of time *on their own* to go through the material as contained in the lab notes and complete the assigned lab work.

Moreover, because of the complexity of the DB applications, it is typically a product of team work. Thus, besides individual programming assignments as found in the lab notes, there will also be a team based project.