

Chapter 1

Open It Up

A *database system* is simply a computerized record keeping system, which is used to manage *databases*. A *database* is a collection of computerized data files related to certain enterprise. Once such databases are set up, a user can apply various operations on them, such as *add* new files into, or *remove* an existing file from, a database; *insert a record into, change, delete, or retrieve, an existing record from,* an existing file in a database.

Database related application is used everywhere all the time. For example, over 100 million credit card transactions are processed each day from over 10 million merchants through more than 20,000 banks.

An example

binno	wine	producer	year	bottles	ready
2	Chardonnay	Buena Vista	1997	1	1999
3	Chardonnay	Geyser Peak	1997	5	1999
12	Joh. Riesling	Jekel	1998	1	1999
21	Fum Blanc	ch. St. Jean	1997	4	1999
22	Fum Blanc	Robt. Mondavi	1996	2	1998
30	gewurztraminer	Ch. St. Jean	1998	3	1999
43	Cab. Sauvignon	Windsor	1991	12	2000
50	Pinot Noir	Gary Farrell	1996	3	1999

A database file such as CELLAR is usually called a *table*, which has *rows*, a bunch of records, and *columns*, a bunch of fields. Every column must have a *data type*. For example, WINE and PRODUCER has the type of character string, and all the rest are of the integer type. Column BIN# is the *primary key* of CELLAR, in the sense that this field uniquely identifies any specific row: no two rows have the same BIN# value.

The vast majority of DBMS are *relational*, or RDBs. Thus, we will mainly focus on the RDBs.

Play with it

The only reason to keep data there is to use them later, e.g., get them out, or more formally, data retrieval. More specifically, if we apply the following SQL *query* to the database:

```
mysql> Select WINE, BINNO, PRODUCER
-> From cellar
-> Where READY=2000;
```

```
+-----+-----+-----+
| WINE          | BINNO | PRODUCER      |
+-----+-----+-----+
| Cab. Sauvignon |    43 | Windsor       |
| Pinot Noir     |    51 | Fetzer        |
| Merlot         |    58 | Clos du Bois  |
+-----+-----+-----+
```

```
3 rows in set (0.05 sec)
```

More queries

Other *update* operations, such as insertion, deletion, and revision, can also be applied. The basic syntactic structure stays the same.

For example, the following adds in another row.

```
Insert into cellar  
(BINNO,WINE,PRODUCER,YEAR,BOTTLES,READY)  
Values (53,'Pinot Noir','Saintsbury',1997,6,2001);
```

We now do it with MySQL:

```
mysql> Insert into cellar  
-> (BINNO,WINE,PRODUCER,YEAR,BOTTLES,READY)  
-> Values (53,'Pinot Noir','Saintsbury',1997,6,2001);  
Query OK, 1 row affected (0.08 sec)
```

We can repeatedly add in more rows...

... until it looks like the following:

```
mysql> select * from cellar;
```

binno	wine	producer	year	bottles	ready
2	Chardonnay	Buena Vista	1997	1	1999
3	Chardonnay	Geyser Peak	1997	5	1999
6	Chardonnay	Simi	1996	4	1998
12	Joh. Riesling	Jekel	1998	1	1999
21	Fum Blanc	ch. St. Jean	1997	4	1999
22	Fum Blanc	Robt. Mondavi	1996	2	1998
30	gewurztraminer	Ch. St. Jean	1998	3	1999
43	Cab. Sauvignon	Windsor	1991	12	2000
45	Cab. Sauvignon	Geyser Peak	1994	12	2002
48	Cab. Sauvignon	Robt. Mondavi	1993	12	2004
50	Pinot Noir	Gary Farrell	1996	3	1999
51	Pinot Noir	Fetzer	1993	5	2000
52	Pinot Noir	Dehlinger	1995	2	1998
53	Pinot Noir	Saintsbury	1997	6	2001
58	Merlot	Clos du Bois	1994	9	2000
64	zinfandel	Cline	1994	9	2003
72	zinfandel	Rafanelli	1995	2	2003

```
17 rows in set (0.00 sec)
```

Notice that the row with its binno being 53 has been added into the table.

Other queries

The following query does an update.

```
mysql> Update cellar
      -> Set BOTTLES=4
      -> Where BINNO=3;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Let's check it out.

```
mysql> select binno, wine, bottles
      -> from cellar where binno=3;
+-----+-----+-----+
| binno | wine          | bottles |
+-----+-----+-----+
|      3 | Chardonnay   |        4 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

A more realistic one

To build meaningful database applications, particularly, transaction processing applications, the power of a full-fledged high-level programming language, such as Java, C++, or PHP, is needed. Moreover, in today's WEB age, lots of database programming are done over the Internet, it thus makes sense to use WEB as a user interface.

We will discuss in the lab part *PHP*, a programming language, and its relationship with *MySQL*, a “free” DBMS, and their connection with *HTML*. The three products are often used together because of they are free, and are reasonably good.

Let's look at an example, `newsletter_signup.html`, and explore the code later in the lab.

What is a database?

Database is a collection of data central to some enterprise, e.g., bank accounts for a bank, student files for a college, personnel files for HR offices. The one we are interested are indeed stored in bits and bytes in a computer. Such a database can be either *centralized* or *distributed* over a widely separated area.

Database related applications are increasingly important for the very existence of a company since it contains mission critical data for that company. For example, for on-line shopping, it provides the only record of enterprise activity. It is also an asset in its own right, when a credit card company use it to check your credit history.

It has also become much more accessible because of the technological progress.

What is a DBMS?

A *Database Management System* (DBMS) is a collection of programs that manages a database.

It supports a high-level access language (e.g. SQL) which a programmer uses to describe (DDL) and get access to (DML) a database.

The description part is usually easy, while the DML part is often put into two classes: *query* and *update*, as we have seen.

A database application is essentially a mixture of various database accesses using an access language (SQL, PHP, etc.). When a query is submitted to a DBMS, it will interpret such a query and then perform requested database access.

Time is money

Some of the databases are timing sensitive, since the real world that those data models changes constantly. Hence, databases often store information about their current state, such as the current balance of your bank account.

When an event happens in the real world that changes such a state, a corresponding change must be made to the relevant information kept in the database to reflect such a change.

For example, when you make a deposit or withdraw, your balance will be changed as well. Particularly, the change corresponding to a withdraw operation needs to be done immediately, i.e., in real time.

What is a transaction?

A *transaction* is a collection of those changes made in real time. Besides changing the state of the database, a transaction might trigger other events, such as sending out cash via an ATM, or dial 911 when it identifies a stolen bank card.

As we will see later, a transaction is atomic, in the sense that either everything in the collection will be done, or nothing is.

For example, when you will withdraw some money from the bank, you will fill a form, give it to the teller, who will then give you the money. If you change your mind, not only you will not get the money, the slip will be shredded as well.

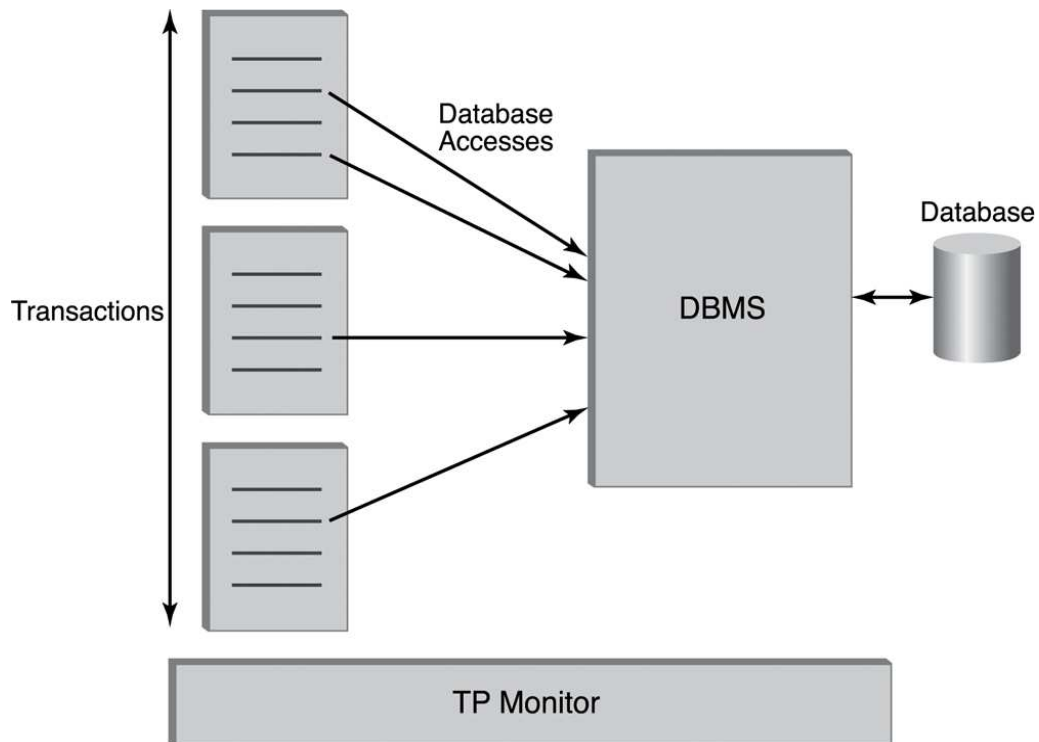
Transaction processing system

Such a system contains a few relevant databases that store the state of the enterprise, the software that manages the transactions that manipulate that state, and the transactions themselves.

When several DBMS are involved, a transaction execution is controlled by a *TP monitor* that is to coordinate transactions across multiple sites.

In many cases, the transaction processing system has become the lifeblood of a company. The credit card business is certainly one of the first examples we can think of.

Worth how many words?



We are mainly interested in the basics of database design and manipulation, but not the algorithms and data structures behind the transaction processing systems.

These topics are often the subjects of a second course on database systems.

Some of the DBMS features

Technological progress has led to significant advances in terms of architecture, design and use of databases and transaction processing systems. They are much faster, can handle much more data, much easier to use and implement, thus providing much more business opportunities. As a price to pay, it also leads to more requirement. For example,

High availability: Since it is now on-line, such systems must be constantly operational. Visa reports in 2002 that their system is down for a total of 8 minutes for five years.

High reliability: The system must be accurate, correctly tracks state, does not lose data, with controlled concurrency.

High throughput: With many users, the system must have the ability to process as many transactions per second.

Low response Time: Users are waiting.

Long lifetime: These systems are not pieces of cake. We have to make lots of efforts and investment. We certainly want them to stay there for a while.

Security: Lots of data are either mission critical or private, thus have to be protected all the time. (In the recent TJ Maxx identity theft case, a hacker stole information of 45.7 million credit cards.)

To keep or not to keep: Even though they are protected, sometimes, they could be subpoenaed such as the recent Google case. It is certainly a different issue.

Major players

A transaction processing system and the involved databases are highly complicated stuff that requires the participation of many people for its success, such as *system analysts* to find out what the user really wants and comes up with a specification for the system; *database designers* to further specify the database structures that keeps the state and supports the needed queries and updates; *application programmers* that implements the GUI part and all the appropriate transactions in the system.

Project managers are responsible for the successful completion of such a project.

Any such a system must be accepted by its *Users*. One of the key points is certainly the user interface, which must be appropriate for their capabilities. To a user, *the interface is the application*.

A *database administrator* maintains database once system is operational in terms of space allocation, performance optimization, database security, etc; (Starting around \$60,000 according to www.payscale.com)

A *system administrator* maintains transaction processing system by monitoring interconnection of hardware and software modules, deals with failures and congestion.

Other stuff

Decision support system is another example where database plays a central role. While a transaction system is to use database to maintain an accurate model of some real world situation; a *decision support system* is to use a database to guide management decisions.

For example, each local supermarket maintains a database that keeps the price and inventory of all the items it sells; while the managers of the chain will have to analyze the local databases to help them make the decision for the chain as a whole, e.g., *what* are selling, and *where*, so that more can be sent there.

In general, we use a on-line analytic processing (OLAP) system to make analysis of information in a database for the purpose of making management decisions.

Data warehouse

Such a system can analyze a large quantity of historical data (terabytes) using complex queries with lots of statistical analysis.

Due to volume of data and complexity of queries, OLAP often uses a *data warehouse*, e.g., sales and inventories from various stores in the last 10 years, taken from OLTP or other sources at various times, perhaps updated once a day.

With such data, a manager can enter a complicated query, e.g., “During the winter months of the last five years, what is the percentage of customers in the Northeast urban supermarkets who bought crackers while buying soup?”

Data mining

A manager can also use warehouse data to discover relationships that might influence enterprise strategy. For example, “Are there any interesting purchasing patterns of our customers?”

Instead of asking for specific information, data mining tries to discover some knowledge in a general nature. It is complicated, but sometimes leads to nice results. For example, a convenience store used the above query and found out that in the early evenings, a high percentage of male customers who bought diapers also bought beers.

Questions: What should we do?

An answer: Put beers next to the diapers?