

Lecture notes by  
Brendan McKay

## Database Management Systems (DBMS)

A DBMS is a software layer that interfaces between the actual data and application programs that wish to use the data. Features:

- The actual structure and location of the data is known only to the DBMS, and only the DBMS needs to know how to access it.
- The DBMS can provide a simplified view of the data to application programs. It might have little similarity to the actual structure of the data in the files.

A **database system** is a collection of data together with a software interface that provides access to the data.

- Application program (possibly interactive).  
↓
- Database Management System (DBMS).  
↓
- Actual data (usually stored in one or more files).

## Database Management Systems (2)

DBMS features, continued:

- The DBMS can provide an extended view of the data (eg. it can collect statistical information about the data that is not actually stored anywhere.)
- The DBMS might provide **security & privacy, transaction logging, checkpointing, locks, atomic transactions, rollbacks**, etc.

## Database Management Systems (3)

DBMS features, continued:

- In some databases, the data is all kept in one place, even if it is accessed from multiple places.
- Alternatively, the data might be kept in multiple locations. This makes it a **distributed database**. Perhaps each item of data is kept in only one place, or items might be kept in multiple places.

### Example: library holdings database

A library holdings database will contain information about **books** held (author, title, ISBN, call number, date acquired, whether it is out on loan), and about library **users** (name, staff or student, starting date, ending date).

It will also contain information about book borrowing (which book, which user, when borrowed, when due).

## Database Models

A database **model** is an abstract view of the way in which the data is organized. The model refers to the view that the DBMS presents to application programs. It does not necessarily have anything to do with the manner in which the data is actually stored.

- Relational.
- Object-oriented.
- Hierarchical.

### Relational model

In the relational model, the data appears to be in the form of tables. Each table is called a **relation**.

**Relation "Books"**

| Author       | Title             | ISBN       | Acquired  | Out |
|--------------|-------------------|------------|-----------|-----|
| Smith, Brian | Database Theory   | 1400906278 | 11-2-2005 | N   |
| June, May    | Database Practice | 1550906278 | 13-6-2006 | N   |

**Relation "Borrowing"**

| ISBN       | Borrower | date out | date in   |
|------------|----------|----------|-----------|
| 1550906278 | u7651412 | 8-5-2007 | 8-8-2007  |
| 1400906278 | u7561123 | 3-3-2007 | 2-10-2007 |

Similarly relation **"Users"**. Each row is called a **tuple**.

## Standard operations in relational model

- SELECT - make a new relation by selecting rows from an existing relation
- PROJECT - make a new relation by selecting columns from an existing relation
- JOIN - make a new relation by combining two existing relations

## Design of relational database schema

The efficiency and robustness of a relational database depends a lot on the choice of relations. Issues include:

- How many tuples need changing in order to perform common changes?
- How much replication of data occurs in different relations?

## Object-oriented model

There are a set of **objects** of different **types**, such as **Book** and **User**.

Objects then have links to each other according to the relationships between them.

An object of type **Book** might link to an object of type **User** that is the current borrower of the book.

An object of type **User** might link to all the objects of type **Book** that this user has currently borrowed.

Objects have **methods** that provide information about those objects and operations involving those objects.

## Hierarchical (tree-structured) model

In this database model, there is a tree structure.

- The root of the tree might be “library” .
- The children of “library” might be the books in the library.
- The children of a book might be the borrower of the book.
- The children of a user might be the books borrowed by the user.

Note how the tree can be infinite.

## Schemas

Irrespective of the database model, a **schema** is a description of the data in the database as provided by the DBMS. For example, in the relational model the schema will list the relations, and for each relation specify what the columns are.

A DBMS might provide a different view to different users. For example, unauthorised users might be forbidden access to sensitive relations or columns. This can be achieved by providing such users with a **subschema** containing only the authorised data.

## Transaction processing

A **transaction** is an operation involving a database, possibly resulting in a change to the data.

Internally, a transaction might involve a sequence of smaller changes, such as modifying fields in several relations.

The DBMS is responsible for managing transactions in such a way that the application programs always see the database in a consistent state.

## Database query languages

A **database query language** is a language that can be used to communicate with a DBMS. There are many.

The most famous database query language is SQL, used with the relational model.

```
SELECT Title, Acquired
FROM Books
WHERE ISBN = "1400906278"
```

## Atomic transaction sequences and locking

For example, an operation might require changes to two different relations. Other operations involving those relations should wait until both changes are made.

One way is for the DBMS to **lock** the relations until the transaction is complete.

Locking is especially difficult to implement efficiently on very large databases and many techniques have been invented to bypass it or make it more efficient.

## Commit and roll back

The moment when all the steps of a transaction have been recorded in the log (before they are executed on the database) is called the **commit point**.

The DBMS then takes responsibility for actually performing the transaction, and meanwhile pretends that it has.

If problems occur, the details in the log can (hopefully) be used to restore the database to its previous state.

This is called **roll back**.

## Security and privacy

Modern DBMSs allow items of data to be classified into **security classes** and application programs to have different **access levels**.

This allows selective presentation of data to different users, to implement safety and privacy policies.

However, thorough protection of privacy is very difficult, even when only statistical information is provided. This is an active area of research.

## Database errors

A major problem with large databases is fault-tolerance. Programs crash, disks have faults, network connections die, etc..

The worst problem is **inconsistency**.

**Transaction logging** is one common technique.

**Checkpointing** refers to saving a complete snapshot of the database at some moment, to enable partial recovery from disasters.

## File structures for databases

Databases frequently contain very large amounts of data, and the speed at which particular items can be found is crucial to the performance.

Many algorithms have been devised for this problem, such as:

- Hash files.
- B-tree files.

## Hash files

Suppose we want to store the Books relation in a file, so that the tuple for a particular ISBN can be found quickly. Say there are 1000 books and each tuple needs 500 bytes (one “slot”).

Make a file of 2003 slots, numbered  $0, 1, \dots, 2002$ .

Store the tuple for ISBN  $N$  in slot  $N \bmod 2003$ .

Eg.  $1400906278 \bmod 2003 = 66 \rightarrow$  use slot 66.

The problem is that some different ISBNs will “hash” to the same slot. There are several methods for handling these **collisions**.

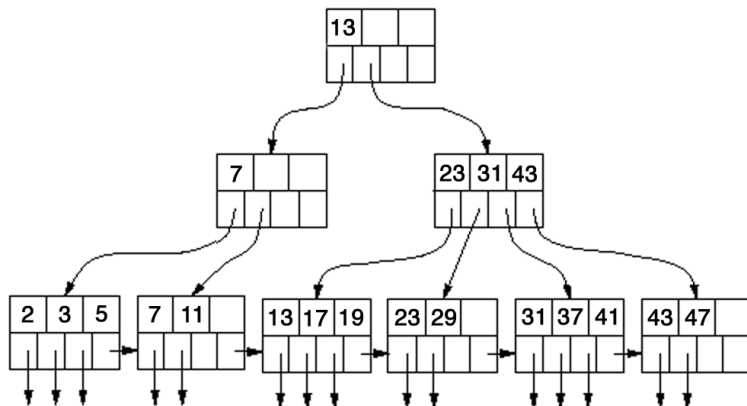
## Hash files (2)

For hash files on disk (unlike those in memory), the main technique for handling collisions is use of **buckets**. Instead of using 2003 slots on disk, we can use (say) 503 buckets with each having space for 4 tuples.

The advantage is that fetching a bucket from disk costs about the same as fetching one slot. Collisions only matter when buckets become full, which happens less often than collisions with individual slots. Tuples hashing to a full bucket are put into **overflow buckets**.

## B-tree files

B-trees allow items to be found with a fairly small number of disk accesses. This example is a “B<sup>+</sup>-tree”.



## B-tree files (2)

In each node, some number of keys are stored. Records with lesser key are stored in leaves to the left and others to the right.

When nodes become full, they are split into two.

Node sizes are chosen according to disk characteristics (say 50-200 keys per node).

Then all records are within a few nodes of the root.