Ian Kenny

September 27, 2016

Databases

Lecture 1

Introduction to the module

Staff: Ian Kenny, room 232, i.s.kenny@cs.bham.ac.uk

Introduction to the module

The purpose of this module is to provide an introduction to topics relating to the design and use of Database Management Systems (DBMSs).

The proposed content is

- Introduction to Database Management Systems
- SQL
- Relational Algebra
- ER modelling
- Logical design: ER diagrams to SQL
- Design theory of DBs: functional dependencies, normalisation
- Performance
- Transaction management

Module structure

There are two lectures per week

- Monday, 1600, Mechanical and Civil Engineering, G34
- Friday, 1000, Mechanical and Civil Engineering, G29

There is one lab session/tutorial per week in the CS labs at **1200 on Fridays**. This is a change from the published timetable.

Module assessment

There are three tests during the module. These are worth 20% of the overall marks.

There is a final exam which is worth 80% of the marks.

The tests will be held in the lecture sessions and will be announced one week in advance.

In this lecture

- Introduction to DBMSs.
- Background theory: the relational model, keys.
- Introduction to SQL
- SQL queries.

Introduction to DBMSs

Database Management Systems are ubiquitous. They sit behind all manner of different systems such as banking, retail, education, government, science, etc. The world would grind to a halt (and worse ...) without DBMSs.

The purpose of a DBMS is to manage very large amounts of data, usually on behalf of other applications. The management of large amounts of data requires specialist processes that it would be expensive, inefficient and wasteful to integrate into applications. The DBMS provides these data management services to applications (hence users).

Introduction to DBMSs

One might wonder why we don't just store data in text files and spreadsheets[1] and write our own code to use the data. However, when an application needs to access that data often, needs to search through that data, needs to combine disparate parts of the data to create desired results, needs to offer multiple users simultaneous access to the data (perhaps users who are distributed across the world), and needs to protect the data from corruption, incomplete processing, and unauthorised and malicious users, the task starts to seem less attractive.

The purpose of a DBMS is to offer this service to applications.

[1] lots of data is stored in this way, in fact.

Goals of a DBMS

- **Scale**: DBMSs are able to work with massive amounts of data: an amount of data that would be impractical without some form of structured storage. Computer system memories are no match for the amount of data that needs to be stored in the modern computing environment.
- **Persistence**: Data is stored and is not lost when the programs using it quit. The data may be used by multiple programs and they need to be able to rely on the data not disappearing when they quit. Whatever gets loaded into RAM will disappear but this is only a temporary copy of that data.

Goals of a DBMS

- **Safety**: DBMSs ensure that hardware failures, software failures, power failures, etc. do not cause the loss of or invalidate the data stored (there may be some minor loss). The integrity of the data should not depend on these external factors. DBMSs also attempt to ensure that unauthorised and malicious use of the data is avoided.
- **Multiple users**: DBMSs allow multiple users using the same applications and different applications to use the same data at the same time without the data's integrity being lost. Users may be distributed geographically and work in different time zones and on different systems. The DBMS manages this on behalf of applications.

Goals of a DBMS

- **Convenience**: DBMSs provide convenient methods for the data to be created, accessed and queried. This is usually through supporting the high-level query language SQL. SQL provides a standard language for operating on the data in a database. This is preferable to each DBMS having its *own* query language or to applications having to query the data directly (which would also break the insulation layer provided by the DBMS between the data and applications).
- **Efficiency**: DBMSs must offer the ability to store and process massive amounts of data whilst not compromising on efficiency and speed. Applications need to be able to query the database in a timely fashion, particularly bearing in mind that many thousands of users could be using a single application to query the database, and that thousands of applications might be querying it.
- **Reliability**: user applications must be able to rely on the DBMS remaining 'up' for virtually 100% of the time. Banking applications, for example, cannot afford even minimal downtime

ACID

In general, a DBMS should embody four properties, known by the acronym *ACID*.

- **Atomicity**: A database *transaction* is a logical unit of processing in the database. For example, a bank transaction that transfers money from one account to another. Such a transaction has many components and should not fail part way through. The Atomicity requirement states that all of a transaction should succeed or none of it.
- **Consistency**: Applications (hence users) should not be able to perform transactions that leave the data in an inconsistent state through the violation of constraints, etc. Each transaction must change the data from one valid state to another valid state.
- **Isolation**: Concurrent operations on the database should lead to the same state changes as if those operations were performed serially.
- **Durability**: Once an operation (for example an SQL statement) has completed it must be durable, i.e. not lost through hardware, software or power faults.

Types of DBMS

We are going to consider the *relational* DBMS since it is by far the most popular commercial type of DBMS.

There are other historical database systems such as hierarchical, object-oriented and network DBMSs. These are not widely implemented and used.

More recently NoSQL databases have become more popular as a means to deal with very large datasets. As their name suggests, these database systems may not use SQL ('No' can stand for 'no' or 'not only') in order to process data. NoSQL databases may not embody the goals of regular DBMSs, as outlined above.

The relational model

In the relational model, data is organised into *relations*. Relations are also called 'tables'. A relation usually represents a logical entity or relationship of some kind, for example, 'Employee'.

A relation has *attributes*. Attributes are also called 'fields' or 'columns'. The attributes of a relation are the 'categories' of information stored about the entity or relationship, for example, 'salary', 'date of birth').

Attributes have a *domain*, i.e. the range of values that the value is taken from. For example, 'strings of length 255 characters', integers, dates.

A relation contains *tuples* (if it is not empty). Tuples are also called 'rows', 'instances'. These are the actual 'instances' of the data in the relation (with reference to object-oriented programming, if the relation is the class then the tuples are like the objects).

Keys

A *key* for a relation is a set of attributes that uniquely identifies the tuples in the relation. The set of all attributes of a relation is clearly a key for the relation but keys usually consist of a subset of the attributes and often a single attribute.

A *candidate key* is any key of the relation.

The *primary key* for a relation is key that has been selected from the candidate keys to act as the key for a relation.

A *foreign key* is a set of attributes in a relation that uniquely identifies a row in a *another table*.

Keys are crucial for the operation of databases.