

What the reader will learn:

- The Origins of databases
- Databases of the 1960's and 1970's
- Current mainstream database technologies—relational versus object orientation
- The need for speed in database systems
- Distributed databases and the challenges of transaction processing with distributed data

2.1 Introduction

In Chap. 1 we discussed data as an organisational asset. We saw data, usually in the form of records has been with us since at least ancient Egyptian times. We also so that the big drivers of the need to keep detailed records were trade and taxation. Basically you needed to keep track of who owed you how much for what. This meant you not only needed a way of recording it, you also needed a way of retrieving it and updating it. Ultimately this lead to the development of double entry book keeping which emerged in the 13th and 14th centuries. Retrieving data was another issue and in paper based systems indexes were developed to ease and speed this process. Producing reports from data was manual, time consuming and error prone although various types of mechanical calculators were becoming common by the mid nineteenth century to aid the process. Because of the time taken to produce them, major reports were usually published once a month coinciding reconciliation of accounts. An end of year financial position was also produced. On demand reports were unheard of. As a result the end of the month and the end of the year were always extremely busy times in the accounts section of an organisation.

2.2 The Digital Age

The first commercial computer UNIVAC 1 was delivered in 1951 to the US Census Bureau (Fig. 2.1). In 1954 Metropolitan Life became the first financial company to

Fig. 2.1 UNIVAC 1 (original at <http://www.computer-history.info/Page4.dir/pages/Univac.dir/>, last accessed 07/08/2013)



purchase one of the machines. This was a significant step forward giving Metropolitan Life a first mover advantage over other insurance companies. The UNIVAC was followed by IBM's 701 in 1953 and 650 in 1954. At the time it was envisaged by IBM that only 50 machines would be built and installed. By the end of production of the IBM 650 in 1962 nearly 2,000 had been installed. Data could now be processed much faster and with fewer errors compared with manual systems and a greater range of reports which were more up to date could be produced. One unfortunate side effect was regarding things as 'must be right, it came from the computer' and conversely 'I can't do anything about it, it is on the computer'. Fortunately both attitudes are no longer as prevalent.

2.3 Sequential Systems

Initial computer systems processing data were based on the pre-existing manual systems. These were sequential systems, where individual files were composed of records organised in some predetermined order. Although not a database because there was no one integrated data source, electronic file processing was the first step towards an electronic data based information system. Processing required you to start at the first record and continue to the end. This was quite good for producing payroll and monthly accounts, but much less useful when trying to recall an individual record. As a result it was not uncommon to have the latest printout of the file which could be manually searched if required kept on hand until a new version was produced. Printing was relatively slow by today's standards, so usually only one copy was produced although multi part paper using interleaved carbon paper was used sometimes when multiple copies could be justified. Sequential file processing also meant there had to be cut offs—times when no new updates could be accepted before the programs were run.

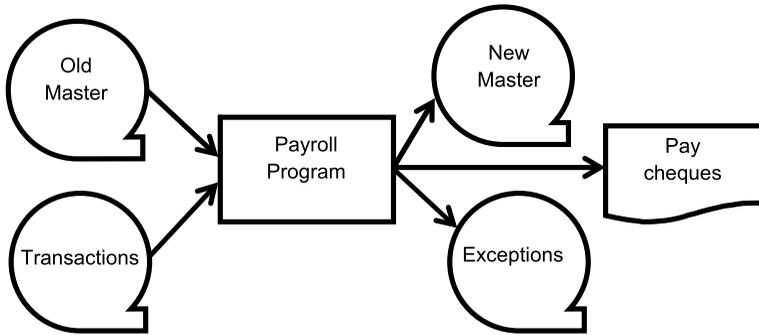


Fig. 2.2 A tape based system

Although disk storage was available it was expensive so early systems tended to be based on 80 column punched cards, punched paper tape or magnetic tape storage, which meant input, update and output had to be done in very specific ways. IBM used card input which was copied onto magnetic tape. This data had to be sorted. Once that was done it could be merged with and used to update existing data (Fig. 2.2). This was usually held on the ‘old’ master tape, the tape containing the processed master records from the previous run of the system (when it was designated the ‘new’ master). Once updated a new master tape was produced which would become the old master in the next run. Processing had to be scheduled. In our example, the first run would be sorting, the second processing. Since only one program could be loaded and run at a time with these systems demand for computer time grew rapidly. As well as data processing time was needed to develop, compile and test new computer programmes. Ultimately many computer systems were being scheduled for up to 24 hours a day, 7 days a week.

There were also very strict rules on tape storage so if a master tape were corrupted it could be recreated by using archived old masters and the transaction tapes. If there were any issues with one of the transactions, it would not be processed, but would be copied to an exception file on another tape. This would be later examined to find out what had gone wrong. Normally the problem would be rectified by adding a new transaction to be processed at the next run of the program.

2.4 Random Access

IBM introduced hard disk drives which allowed direct access to data in 1956, but these were relatively low capacity and expensive compared to tape system. By 1961 the systems had become cheaper and it was possible to add extra drives to your system. The advantage of a disk drive was you could go directly to a record in the file which meant you could do real time transaction processing. That is now the basis of

processing for nearly all commercial computer systems. At the same time operating systems were developed to allow multiple users and multiple programs to active at the same time removing some of the restrictions imposed by tight scheduling needed on single process machines.

2.5 Origins of Modern Databases

It was inevitable that data would be organised in some way to make storage and retrieval more efficient. However, while they were developing these systems there was also a move by the manufacturers to lock customers into their proprietary products. Many failed forcing early adopters of the ‘wrong’ systems to migrate to other widely adopted systems, usually at great expense. For example Honeywell developed a language called FACT (Fully Automated Compiler Technique) which was designed for implementing corporate systems with associated file structures. The last major user was the Australian Department of Defence in the 1960’s and 70’s. They took several years and a large budget to convert to UNIVAC’s DMS 1100 system which will be described below.

UNIVAC and IBM competed to develop the first database where records were linked in a way that was not sequential. UNIVAC had decided to adopt the COBOL (Common Business Oriented Language) programming language and therefore also adopted the CODASYL (CONference on DATA SYstems Languages) conventions for developing their database. CODASYL was a consortium formed in 1959 to develop a common programming language which became the basis for COBOL. Interestingly despite Honeywell being a member of the CODASYL group, they tried to put forward their FACT language as tried and functioning alternative to the untried COBOL. As shown this strategy was not successful. In 1967 CODASYL renamed itself the Database Task Group (DBTG) and developed a series of extensions to COBOL to handle databases. Honeywell, Univac and Digital Equipment Corporation (DEC) were among those who adopted this standard for their database implementations.

IBM on the other hand had developed its own programming language PL/1 (Programming Language 1) and a database implementation known as IMS (Information Management System) for the extremely popular IBM Series 360 computer family. The UNIVAC system was a network database and the IBM a strictly hierarchical one. Both were navigational systems where you accessed the system at one point then navigated down a hierarchy before conducting a sequential search to find the record you wanted. Both systems relied on pointers.

The following sections will look at transaction processing which underlies many database systems, then different database technologies will be briefly examined in roughly the order they appeared. Many of these will be dealt with in more detail in later chapters.

2.6 Transaction Processing and ACID

Many database applications deal with transaction processing, therefore we need to look at what a transaction is. Basically a transaction is one or more operations that make up a single task. Operations fall into one of four categories; Create, Read, Update or Delete (so called CRUDing). As an example you decide to make a withdrawal at a cash machine. There are several transactions involved here, first you need to be authenticated by the system. We will concentrate on the actual withdrawal. The system checks you have enough money in your account (Read); it then reduces the amount in your account by the amount requested (Update) and issues the money and a receipt. It also logs the operation recording your details, the time, location and amount of the withdrawal (Create). Several things can happen that cause the transaction to abort. You may not have enough money in the account or the machine may not have the right denomination of notes to satisfy your request. In these cases the operations are undone or rolled back so your account (in other words the database) is returned to the state it was in before you started.

ACID stands for atomicity, consistency, isolation and durability and is fundamental to database transaction processing. Earlier in this chapter we saw how transactions were processed in sequential file systems, but with multiuser systems which rely on a single database, transaction processing becomes a critical part of processing. Elements of this discussion will be expanded on in the chapters on availability and security.

Atomicity refers to transactions being applied in an ‘all or nothing’ manner. This means if any part of a transaction fails, the entire transaction is deemed to have failed and the database is returned to the state it was in before the transaction started. This returning to the original state is known as rollback. On the other hand, if the transaction completes successfully the database is permanently updated—a process known as commit.

Consistency means data written to a database must be valid according to defined rules. In a relational database this includes constraints which determine the validity of data entered. For example, if an attempt was made to create an invoice and a unassigned customer id was used the transaction would fail (you can’t have an invoice which does not have an associated customer). This would also trigger the atomicity feature rolling back the transaction.

Isolation ensures that transactions being executed in parallel would result in a final state identical to that which would be arrived at if the transactions were executed serially. This is important for on line systems. For example, consider two users simultaneously attempting to buy the last seat on an airline flight. Both would initially see that a seat was available, but only one can buy it. The first to commit to buying by pressing the purchase button and having the purchase approved would cause the other users transaction to stop and any entered data (like name and address) to be rolled back.

Durability is the property by which once a transaction has been committed, it will stay committed. From the previous example, if our buyer of the last seat on the aeroplane has finished their transaction, received a success message but the system crashes for whatever reason before the final update is applied, the user would reasonably think their purchase was a success. Details of the transaction would have to be stored in a non-volatile area to allow them to be processed when the system was restored.

2.7 Two-Phase Commit

Two-phase commit is a way of dealing with transactions where the database is held on more than one server. This is called a distributed database and these will be discussed in more detail below. However a discussion on the transaction processing implications belongs here. In a distributed database processing a transaction has implications in terms of the A (atomicity) and C (consistency) of ACID since the database management system must coordinate the committing or rolling back of transactions as a self-contained unit on all database components no matter where they are physically located. The two phases are the request phase and the commit phase. In the request phase a coordinating process sends a query to commit message to all other processes. The expected response is 'YES' otherwise the transaction aborts. In the commit phase the coordinator sends a message to the other processes which attempt to complete the commit (make changes permanent). If any one process fails, all processes will execute a rollback.

Oracle, although calling it a two-phase commit has defined three phases: prepare, commit and forget which we will look at in more detail. In the prepare phase all nodes referenced in a distributed transaction are told to prepare to commit by the initiating node which becomes the global coordinator.

Each node then records information in the redo logs so it can commit or rollback. It also places a distributed lock on tables to be modified so no reads can take place. The node reports back to the global coordinator that it is prepared to commit, is read-only or abort. Read-only means no data on the node can be modified by the request so no preparation is necessary and will not participate in the commit phase. If the response is abort it means the node cannot successfully prepare. That node then does a rollback of its local part of the transaction. Once one node has issued a abort, the action propagates among the rest of the nodes which also rollback the transaction guaranteeing the A and C of ACID.

If all nodes involved in the transaction respond with prepared, the process moves into the commit phase. The step in this phase is for the global coordinator node to commit. If that is successful the global coordinator instructs all the other nodes to commit the transaction. Each node then does a commit and updates the local redo log before sending a message that they have committed. In the case of a failure to commit, a message is sent and all sites rollback.

The forget phase is a clean-up stage. Once all participating nodes notify the global coordinator they have committed, a message is sent to erase status information about the transaction.

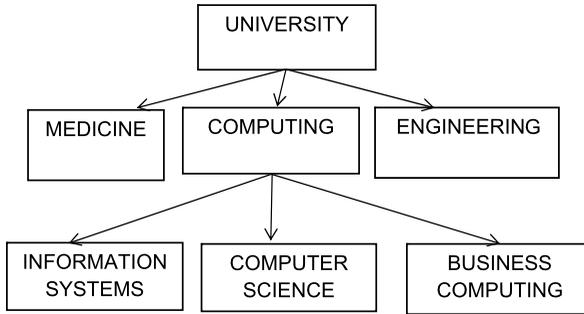


Fig. 2.3 Hierarchical database

SQLServer has a similar two phase commit strategy, but includes the third (forget) phase at the end of the second phase.

2.8 Hierarchical Databases

Although hierarchical databases are no longer common, it is worth spending some time on a discussion of them because IMS, a hierarchical database system is still one of IBM's highest revenue products and is still being actively developed. It is however a mainframe software product and may owe some of its longevity to organisations being locked in to the product from the early days of mainstream computing. The host language for IMS is usually IBM's PL/1 but COBOL is also common.

Like a networked databases, the structure of a hierarchical database relies on pointers. A major difference is that it must be navigated from the top of the tree (Fig. 2.3).

2.9 Network Databases

In a network database such as UNIVAC's DMS 1100, you have one record which is the parent record. In the example in Fig. 2.4 this is a customer record. This is defined with a number of attributes (for example Customer ID, Name, Address). Linked to this are a number of child records, in this case orders which would also have a number of attributes. It is up to the database design to decide how they are linked. The default was often to 'next' pointers where the parent pointed to the first child, the first child had a pointer to the second child and so on. The final child would have a pointer back to the parent. If faster access was required, 'prior' pointers could be defined allowing navigation in a forward and backward direction. Finally if even more flexibility was required 'direct' pointers could be defined which pointed directly from a child record back to the parent. The trade-off was between speed of access and speed of updates, particularly when inserting new child records and deleting records. In these cases pointers had to be updated.

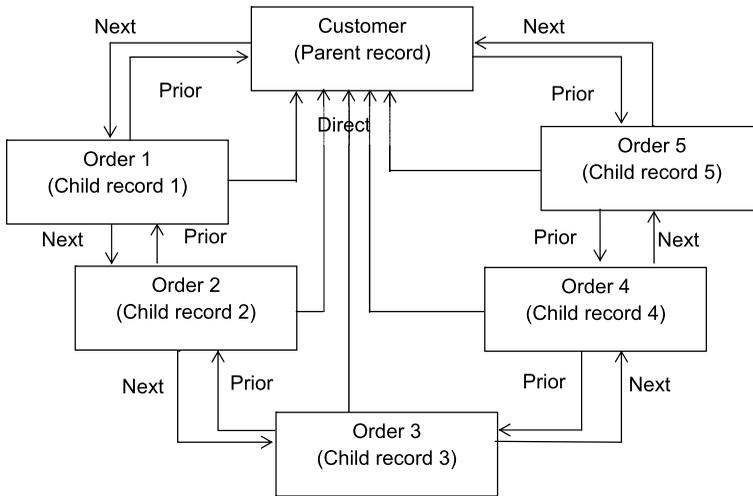


Fig. 2.4 Network database

The whole of the definition of the database and the manipulation of it was hosted by the COBOL programming language which was extended to deal with DMS 1100 database applications. It is the main reason why the COBOL language is still in use today. It is worth noting that deficiencies in the language led to fears there would be major computer (which actually meant database) failure at midnight on 31st December 1999. This was referred to as the Y2K or millennium bug and related to the way dates were stored (it was feared some systems would think the year was 1900 as only the last two digits of the year were stored). The fact that the feared failures never happened has been put down to the maintenance effort that went into systems in the preceding few years.

2.10 Relational Databases

These will be discussed in detail in Chap. 4. They arose out of Edgar Codd’s 1970 paper “A Relational Model of Data for Large Shared Data Banks” (Codd 1970). What became Oracle Corporation used this as the basis of what became the biggest corporate relational database management system. It was also designed to be platform independent, so it didn’t matter what hardware you were using.

The basis of a relational system is a series of tables of records each with specific attributes linked by a series of joins. These joins are created using foreign keys which are attribute(s) containing the same data as another tables primary key. A primary key is a unique identifier of a record in a table. This approach to data storage was very efficient in terms of the disk space used and the speed of access to records.

In Fig. 2.5 we have a simple database consisting of 2 tables. One contains employee records, the other contains records relating to departments. The Department ID can be seen as the link (join) between the two tables being a primary key in the

Employee

Employee ID	Surname	Given Name	Initial	Department ID	DOB	Salary
35411	Felsky	Warren	J	22	25/2/1983	21000.00
35567	Cash	Martin	B	21	1/6/1989	19750.00
35582	Kelly	Edward	T	22	18/7/1961	45450.00



Department

Department ID	Department Name	Department Location
21	Accounts	Head office
22	Payroll	Head office
23	Sales	Latrobe

Fig. 2.5 Relational database consisting of two tables or relations

Department table and a foreign key in the Employee table. There is also a one to many relationship illustrated which means for every record in the Department table there are many records in the Employee table. The converse is also true in that for every record in the Employee table there is one and only one record in the Department table.

The data in a relational database is manipulated by the structured query language (SQL). This was formalised by ANSI (American National Standards Institute) in 1986. There have been seven revisions of SQL86, the most recent revision being SQL2011 or ISO/IEC 9075:2011 (International Organization for Standardization /International Electrotechnical Commission). SQL has a direct relationship to relational algebra. This describes tables (in relational algebra terms—relations), records (tuples) and the relationship between them. For example:

$$P_1 = \bullet_{\text{type_property}=\text{'House'}}(\text{Property_for_Rent})$$

would be interpreted as find all the records in the property for rent table where the type of rental property is a house and would be written in SQL as:

```
SELECT * FROM property_for_rent WHERE type_property = 'House';
```

Oracle and Microsoft SQL server are examples of relational database systems. Microsoft Access has many of the features of a relational database system including the ability to manipulate data using SQL, but is not strictly regarded as a relational database management system.

2.11 Object Oriented Databases

Most programming today is done in an object oriented language such as Java or C++. These introduce a rich environment where data and the procedures and functions need to manipulate it are stored together. Often a relational database is seen by object oriented programmers as a single persistent object on which a number of operations can be performed. However there are more and more reasons why this is becoming a narrow view.

One of the first issues confronting databases is the rise of non-character (alphanumeric) data. Increasingly images, sound files, maps and video need to be stored, manipulated and retrieved. Even traditional data is being looked at in other ways than by traditional table joins. Object oriented structures such as hierarchies, aggregation and pointers are being introduced. This has led to a number of innovations, but also to fragmentation of standards.

From the mid 1980's a number of object oriented database management systems (OODBMS) were developed but never gained traction in the wider business environment. They did become popular in niche markets such as the geographic and engineering sectors where there was a need for graphical data to be stored and manipulated. One of the big issues with object oriented databases is, as mentioned creating a standard. This was attempted by the Object Data Management Group (ODMG) which published five revisions to its standard. This group was wound up in 2001. The function of ODMG has been taken over by the Object Management Group (OMG). Although there is talk of development of a 4th generation standard for object databases, this has not yet happened

An issue with developing object oriented systems was that a lot of expertise and systems had been developed around relational databases and SQL had become, for better or worse, the universal query language. The second approach to object orientation therefore was to extend relational databases to incorporate object oriented features. These are known as object relational databases. They are manipulated by SQL commands which have been extended to deal with object structures.

Chapter 7 will look in detail at both Object and Object Relational Databases.

2.12 Data Warehouse

A problem with an operational database is that individual records within it are continually being updated, therefore it is difficult to do an analysis based on historical data unless you actively store it. The concept behind a data warehouse is that it can store data generated by different systems in the organisation, not just transactions from the database. This data can then be analysed and the results of the analysis used to make decisions and inform the strategic direction of an organisation. Data warehouses have already been mentioned in Chap. 1 where they were discussed in their role of an organisational asset. Here we look briefly at some of their technical details.

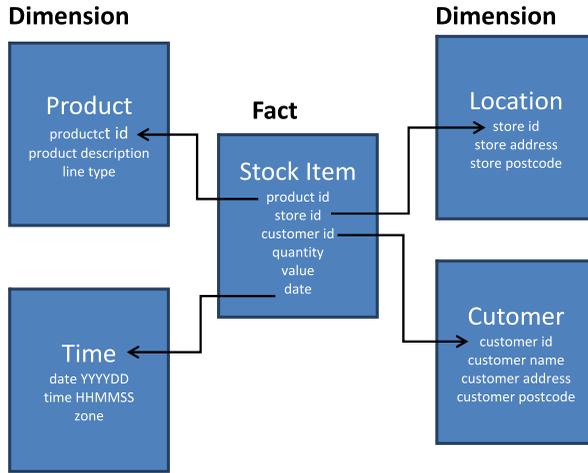


Fig. 2.6 Star schema

A data warehouse is a central repository of data in an organisation storing historical data and being constantly added to by current transactions. The data can be analysed to produce trend reports. It can also be analysed using data mining techniques for generating strategic information. Unlike data in a relational database, data in a data warehouse is organised in a de-normalised format. There are three main formats:

- Star schema: tables are either fact tables which record data about specific events such as a sales transaction and dimension tables which information relating to the attributes in the fact table (Fig. 2.6)
- Snowflake schemas are also based around a fact table which has dimension tables linked to it. However these dimension tables may also have further tables linked to them giving the appearance of a snowflake (Fig. 2.7)
- Starflake schema which is a snowflake schema where only some of the dimension tables have been de-normalised

Data mining is a process of extracting information from a data warehouse and transforming it into an understandable structure. The term data analytics is starting to be used in preference to data mining. There are a number of steps in the process:

- Data cleansing were erroneous and duplicate data is removed. It is not uncommon in raw data to have duplicate data where there is only a minor difference, for example two customer records may be identical, but one contains a suburb name as part of the address and one does not. One needs removing.
- Initial Analysis where the quality and distribution of the data is determined. This is done to determine if there is any bias or other problems in the data distribution
- Main analysis where statistical models are applied to answer some question, usually asked management.

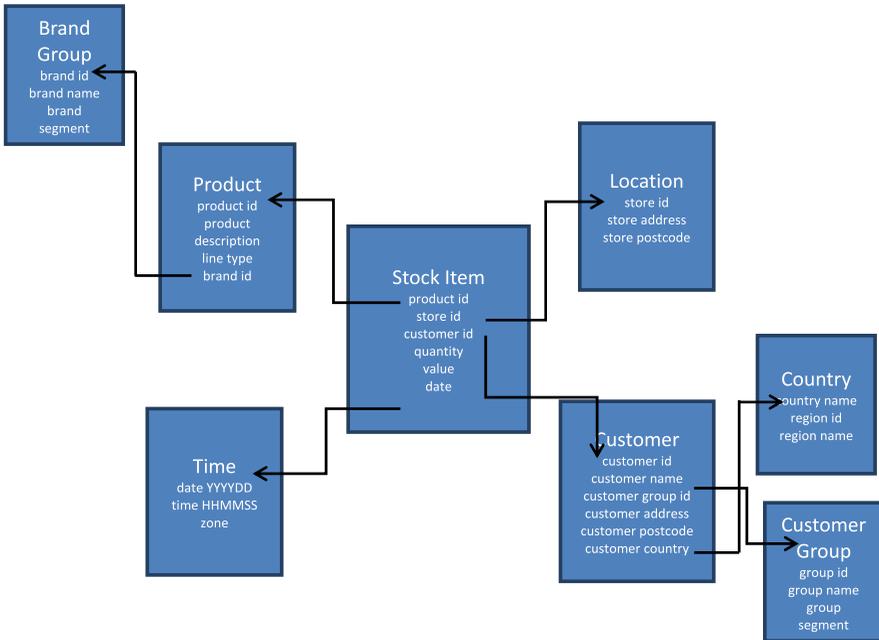


Fig. 2.7 Snowflake schema

Data mining is just one method in a group collectively known as Business Intelligence that transform data into useful business information. Others included business process management and business analytics. A further use for data in a data warehouse is to generate a decision system based on patterns in the data. This process is known as machine learning. There are two main branches: neural networks which use data to train a computer program simulating neural connections in the brain for pattern recognition and rule based systems where a decision tree is generated.

2.13 The Gartner Hype Cycle

Gartner began publishing its annual hype cycle report for emerging technologies in 2007. Technologies such as Big Data and Cloud Computing both feature so we will have a brief look at the hype cycle before continuing. The hype cycle is based on media 'hype' about new technologies and classifies them as being in one of five zones.

- **The Technology Trigger** which occurs when a technological innovation or product generates significant press interest
- **The Peak of Inflated Expectations** where often unrealistic expectations are raised
- **The Trough of Disillusionment** where, because of failing to meet expectations there is a loss of interest

- **The Slope of Enlightenment** where organisations which have persisted with the technology despite setbacks begin to see benefits and practical applications
- **The Plateau of Productivity** where the benefits of the technology have been demonstrated and it becomes widely accepted

In practice it is not really a cycle but a continuum. Gartner has produced many different hype cycles covering many different topics from advertising through to wireless network infrastructure (see Gartner Inc. 2013).

One of the exercises at the end of this chapter involves examining hype cycles. By completing this you will see what they look like and how technologies move across the cycle on a year by year basis.

2.14 Big Data

As mentioned in Chap. 1, the amount of data available that an organisation can collect and analyse has been growing rapidly. The first reference to big data on Gartner's Hype Cycle of emerging technologies was in 2011 when it was referred to as 'Big Data and Extreme Information Processing and Management'. As a term it grew out of data warehousing recognising the growth of storage needed to handle massive datasets some organisations were generating.

The definition of "Big data" is often left intentionally vague, for example McKinsey's 2011 report defines it as referring '...to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze.' (Manyika et al. 2011, p. 1). This is because different organisations have different abilities to store and capture data. Context and application are therefore important. Big Data will be explored in depth in Chap. 6.

2.15 Data in the Cloud

The 2012 Gartner Hype Cycle also identifies database platform as a service (as part of cloud computing) and like in-memory databases it has been identified as over-hyped. However like in-memory databases it is also thought to be within five years of maturity.

Cloud computing is distributed computing where the application and data a user may be working with is located somewhere on the internet. Amazon Elastic Compute Cloud (Amazon EC2) was one of the first commercial cloud vendors supplying resizable computing and data capacity in the cloud. The Amazon cloud is actually a large number of servers located around the world although three of the biggest data centres are located in the United States, each with thousands of servers in multiple buildings. An estimate in 2012 suggested Amazon had over 500,000 servers in total. As a user you can buy both processing and data storage when you need it. Many other vendors including British Telecom (BT) are now offering cloud services.

The impact on database systems is you no longer have to worry about local storage issues because you can purchase a data platform as a service in the cloud. It also means whatever an organisations definition of big data is, there is a cost effective

way of storing and analysing that data. Although concerns have been raised about security including backup and recovery of cloud services, they are probably better than policies and practice in place at most small to medium sized businesses.

2.16 The Need for Speed

One of the technologies facilitating databases was hard disk drives. These provided fast direct access to data. As already stated, the cost of this technology has been falling and speed increasing. Despite this the biggest bottleneck in data processing is the time it takes to access data on a disk. The time taken has three components: seek time which is how long it takes the read/write head to move over the disk and rotational latency, basically the average time it takes the disk to rotate so the record you want is actually under the read/write head. Data then has to be moved from the disk to the processor—the data transfer rate.

For many applications this is not a problem, but for others performance is an issue. For example stockbrokers use systems called High Frequency Trading or micro-trading where decisions are made in real time based on transaction data. This is automated and based on trading algorithms which use input from market quotes and transactions to identify trends. Requirements for speed like this are beyond disk based systems but there are ways to get around the problem. The first is to move the entire database into memory; the second is to use a different access strategy to that provided by the database management system; finally there is a combination of the two.

The 2012 Gartner Hype cycle for emerging technologies identifies in-memory database management systems and in-memory analysis as part of its analysis. However both of these are sliding into what Gartner term the ‘trough of disillusionment’ meaning they have probably been overhyped. On the other hand they are also predicted to be within 5 years of maturity which Gartner calls the ‘plateau of productivity’

2.17 In-Memory Database

Early computer systems tried to optimise the use of main memory because it was both small and expensive. This has now changed with continued development of semi-conductor memory, specifically non-volatile memory. This was a major development as it meant data stored in memory was not lost if there was a power failure. It is now possible to hold a complete database in a computer’s memory. There is no one dominant technology at the moment. Oracle, for example has taken the approach of loading a relational database in its TimesTen system. The problem with this approach is a relational database is designed for data stored on disk and helps optimise the time taken to transfer data to memory. This is no longer an issue as the data is already in memory meaning other ways of organising and accessing data can be implemented. In-memory databases will be looked at in detail in Chap. 8.

2.18 NoSQL

One problem with the relational database model was that it was designed for character based data which could be modelled in terms of attributes and records translated to columns and rows in a table. It has also been suggested it does not scale well which is a problem in a world where everyone is talking about big data. NoSQL (some say Not Only SQL) is an umbrella term for a number of approaches which do not use traditional query languages such as SQL and OQL (Object Query Language). There are a number of reasons for the rise in interest on No SQL. The first is speed and the poor fit of traditional query languages to technologies such as in-memory databases mentioned above. Secondly there is the form of the data which people want to store, analyse and retrieve. Tweets from Twitter are data that do not easily fit a relational or object oriented structure. Instead column based approaches where a column is the smallest unit of storage or a document based approach where data is denormalised can be applied. NoSQL will be discussed in Chap. 5.

2.19 Spatial Databases

Spatial databases form the basis of Geographic Information Systems (GIS). Most databases have some geographic information stored in them. At the very least this will be addresses. However there is much more information available and combining it in many ways can provide important planning information. This process is called overlaying. For example you may have a topographic map of an area which gives height above sea level at specific points. This can then have a soil, climate and vegetation map overlaid on it to.

An issue with spatial databases is that as well as storing ‘traditional’ data you also have to store positional and shape data. As an example, let’s take a road. This can be stored as all the pixels making up the road (know as a raster representation) or it could be stored as a series of points (minimum two if it is a straight road) which can be used to plot its location (vector representation). Along with a representation of its shape and location you may also want to store information such as its name and even information about its construction and condition. As well as plotting this object you may want to know its location in relation to service infrastructure such as sewer lines, for example where the road crosses a major sewer or where access points to it are.

Despite expecting GISs’ to be object oriented, for the most part they are relational. For example the biggest vendor in this field is ESRI which uses a proprietary relational database called ArcSDE. Oracle Spatial is a rapidly emerging player and has the advantage of offering this product as an option to its mainstream database management system (currently 11g). As Oracle Spatial is part of the wider group of Oracle products, it means there are associated object oriented features and a development environment available.

2.20 Databases on Personal Computers

Before 1980 most database systems were found on mainframe computers, but the advent of personal computers and the recognition that they were viable business productivity tools lead to the development of programs designed to run on desktop machines. The first widely used electronic spreadsheet to be developed was VisiCalc in 1979 which spelled a rather rapid end to the paper versions. Word processors and databases followed soon after.

In the early days of personal computers there was a proliferation of manufacturers many of whom had their own proprietary operating systems and software. The introduction of IBM's personal computer in 1980 using Microsoft's operating system resulted in a shake out of the industry. Most manufacturers who remained in the market started producing machines based on the IBM PC's architecture (PC clones) and running Microsoft software. In 1991 the LINUX operating system was released as an alternative to Microsoft's operating system. The largest of the other companies was Apple which vigorously opposed the development of clones of its machines.

There are several databases systems which run on personal computers. One of the earliest was DBase developed by Ashton Tate in 1980. Versions of this ran on most of the personal computers available at the time. The ownership of the product has changed several times. The product itself has developed into an object oriented database whose usage is not common among business users, but is still quite popular among niche users.

Microsoft Access was a much later product being released in 1992. It was bundled with the Microsoft Office suite in 1995 and has become one of the most common desktop databases in use today. One of its attractions is that its tables and relationships can be viewed graphically but it can also be manipulated by most SQL statements. It is however not a true relational database management system because of the way table joins are managed. It is compatible with Microsoft's relational database system SQL Server. One of its major disadvantages is that it does not scale well.

An open source personal computer database is MySQL released in 1995. This is a relational database and is very popular. However although the current open source market leader NoSQL and NewSQL based products are gaining on it according to Matt Asya's in his 2011 article 'MySQL's growing NoSQL problem'. (Available at http://www.theregister.co.uk/2012/05/25/nosql_vs_mysql/ last accessed 04/07/2013.)

2.21 Distributed Databases

In Chap. 1 we saw the evolution of organisations often lead to them being located on multiple sites and sometimes consisting of multiple federated but independent components. Therefore rather than all the data in an organisation being held on a single centralised database, data is sometimes distributed so operational data is held on the site where it is being used. In the case of a relational database this requires tables or

parts of tables to be fragmented and in some case replicated. Early versions of distribution was happening in the mid 1970's, for example the Australian Department of Defence had a centralised database system but bases around Australia replicated segments based on their local needs on mini computers. This was not a real time system, rather the central systems and the distributed machines synchronised several times a day.

Today it is possible to have components or fragments of a database on a network of servers whose operation is transparent to users. There are four main reasons to fragment or distribute databases across multiple sites or nodes. The first is efficiency where data is stored where it is most frequently used. This cuts the overhead of data transmission. Obviously you don't store data that is not used by any local application. This also leads to the second reason—security, since data not required at local nodes is not stored there it makes local unauthorised access more difficult. The third reason is usage. Most applications work with views rather than entire tables, therefore the data required by the view is stored locally rather than the whole table. The final reason is parallelism where similar transactions can be processed in parallel at different sites. Where multiple sites are involved in a transaction a two phase commit process is required and is described in the following section.

There are some disadvantages however. The database management system has to be extended to deal with distributed data. Poor fragmentation can result in badly degraded performance due to constant data transfer between nodes (servers). This may also lead to integrity problems if part of the network goes down. There needs to be a more complex transaction processing regime to make sure data integrity is maintained.

2.22 XML

XML stands for eXtensible Markup Language and is used for encoding a document in a form that is both human and machine readable. It is not intended to give a full description of XML here. What is important from a database point of view is that you can define a database in terms of XML which is important for document databases. There are two forms of XML schema which constrains what information can be stored in a database and constrain the data types of the stored information. It must be noted an XML document can be created without an associated schema, but those will not be considered here.

Document Type Definition (DTD) This is an optional part of a XML document and performs the task of constraining and typing the information in the document. This constraining is not like the basic types of integer and characters (VARCHAR) we have already seen. Instead it constrains the appearance of sub-elements and attributes within an element. An example DTD may look like:

```

<!DOCTYPE account [
  <!ELEMENT invoice (invoice-number total)>
  (!ATTLIST invoice
    account-number ID #REQUIRED
    owner IDREF #REQUIRED>
  <!ELEMENT customer (customer-name customer street customer-city)>
  (!ATTLIST customer
    customer-id ID #REQUIRED
    accounts IDREF #REQUIRED
    :
  ]>

```

where an attribute of type ID must be unique and a IDREF is a reference to an element and must contain a value that appears in an ID element. This effectively gives a one to many relationship.

DTD have limitations however. They cannot specify data types and are written in their own language, not XML. Also they do not support newer XML features such as namespaces. Namespaces are used to contain definitions of identifiers. Once a namespace is set up a developer will be confident of identifiers definitions. The same identifier in a different namespace could have different definitions.

XML Schema This is a more sophisticated schema language designed to overcome the deficiencies of DTD at the same time providing backwards (and forwards) compatibility. It also allows user defined types to be created and text that appears in elements to be constrained to more familiar types such as numeric including specific format definitions. Most importantly it is specified by XML syntax.

Once a database is set up using XML it can be queried using a number of languages:

- XPath is designed for path expressions and is the basis for the following two languages.
- XSLT is a transformation language. It was originally designed to control formatting of XML to HTML but it can be used to generate queries.
- XQuery is the current proposed standard for querying XML data.

It should be noted that many database management systems including Oracle are compatible with XML and can format output and receive input as XML. As mentioned in Chap. 1, there is a tendency to store anything that can be stored. Much of this data is in the form of complete documents. XML databases are an ideal way to store these therefore they are likely to become more and more common.

2.23 Temporal Databases

A temporal database is a database with built-in support for handling data involving time. A lot of data has a time element to it just as we have seen data often has a geographic element to it. Unlike the geographic element however, the time attributes are less often stored. The exception is transaction logs where temporal data is critical.

Two aspects of time are valid time—the time during which a fact is true in the real world and transactional time where it is true in a database. For example when an electricity meter reading is taken, at the instance it is taken it is true in real time. It is then entered into the supplier’s database along with the time the reading was taken. This is the beginning of when it is true in transactional time. It remains true until the time the next reading is entered. It means the reading has start and an end point (the time of the next reading) during which it is true.

2.24 Summary

In this chapter we have looked at the origins of organisational data processing beginning with systems based around sequential file processing. We then looked at the origins and development of database technology as random access devices became cheaper, faster and more reliable. The impact of Edgar Codd’s influential work and the rise to dominance of relational database systems was examined in detail and will be the subject of further parts of this book. Competing with the relational database model is the object oriented model which is gaining traction because the amount of non-structured and non-character based data is increasing.

We looked at developing requirements for databases. New applications demand faster processing of data held in a database. Cheaper, stable computer memory is facilitating this so a database can be held in-memory. As well as a need for speed there is a need for volume as typified by Big Data and the tools need to analyse it. Lastly we looked at distributed databases placing data where it is needed.

2.25 Exercises

2.25.1 Review Questions

The answers to these questions can be found in the text of this chapter.

- What is the difference between a Hierarchical and a Network database?
- Define the following relational database terms: relation, tuple, attribute, relationship.
- What is ACID and why is it important?
- Describe the stages in the two phase commit process.
- Why have object oriented databases not become the standard commercial database management systems?

2.25.2 Group Work Research Activities

These activities require you to research beyond the contents of the book and can be tackled individually or as a discussion group.

Activity 1 Search for the yearly Gartner Hype Cycle on-line. The earliest of these is from 2009. Look for database related technologies. When did they start to appear? From 2011 Gartner also produced Hype for the Cloud. Look at this as well and identify relevant technologies. Are the technologies you identified moving along the hype cycle year on year? When did they first appear on the hype cycle?

The most recent version of the hype cycle is likely to be a document you have to pay for, but earlier versions are freely available.

Activity 2 Retrieve M. Asya's (2011) 'MySQL's growing NoSQL problem' article. (Available at http://www.theregister.co.uk/2012/05/25/nosql_vs_mysql/ last accessed 04/07/2013.) This makes predictions on the growth of MySQL, NoSQL and NewSQL. Research whether Asya's predictions are on track and if there are significant variations what is causing them?

References

- Codd EF (1970) A relational model of data for large shared data banks. Republished in Commun ACM 26(1):64–69 (1983). Available on-line at <http://dl.acm.org/citation.cfm?id=358007>, accessed 21/06/2013
- Gartner Inc (2013) Research methodologies: hype cycles. Available online at <http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>, accessed 21/06/2013
- Manyika J, Chui M, Brown B, Bughin J, Dobbs R, Roxburgh C, Hung Byers A (2011) Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation, accessed 23/06/2013

Further Reading

- ODBMS.ORG (2006) 4th generation standard for object databases on its way. <http://odbms.org/About/News/20060218.aspx>, accessed 23/06/2013
- Özsu MT, Valduriez P (2010) Principles of distributed database systems, 3rd edn. Springer, Berlin. Available at www.stanford.edu/class/cs347/reading/textbook.pdf, accessed 25/06/2013



<http://www.springer.com/978-1-4471-5600-0>

Concise Guide to Databases

A Practical Introduction

Lake, P.; Crowther, P.

2013, XIX, 307 p. 113 illus., Softcover

ISBN: 978-1-4471-5600-0