SKILLS DEVELOPMENT PROJECT
Ministry of Tertiary Education & Training

National Diploma in Information & Communication Technology

Database Management System

Notes

209

Developed by
Interactive Training Division
IDM Computer Studies (Pvt) Ltd.
http://www.idm.edu

CHAPTER 1

Fundamentals of Database Concept

Introduction

In this section we discuss in an informal manner the idea of a database as an abstract machine. An abstract machine is a model of the key features of some system without any details of implementation. The objective of this section is to describe the fundamental concepts of a database system without introducing any formal notation, or introducing any concepts of representation, development or implementation.

What is a Database?

Most modern-day organizations have a need to store data relevant to their day today activities. Those organizations choose an electronic database to organize and store some of this data.

Take for instance a university. Most universities need to record data to help in the activities of teaching and learning. Most universities need to record, among other things:

- What students and lecturers they have
- What courses and modules they are running
- Which lecturers are teaching which modules
- Which students are taking which modules
- Which lecturer is assessing against which module
- Which students have been assessed in which modules

Various members of staff at a university will be entering data such as this into a database system. For instance, administrators in academic departments may enter data relevant to courses and modules, course co-coordinators may enter data pertaining to lecturers, and data relevant to students, particularly their enrolments on courses and modules, may be entered by staff at a central registry.

Once the data is entered into the database it may be utilized in a variety of ways. For example, a complete and accurate list of enrolled students may be used to generate membership records for the learning resources center; it may be used as a claim to educational authorities for student income or as an input into a timetabling system which might attempt to optimize room utilization across a university campus.

In this section our objective is to learn fundamental concept of a database system using this example of an academic database to illustrate concepts.

Properties of a Database

The term database also usually implies a series of related properties: data sharing, data integration, data integrity, data security, data abstraction and data independence.

Data sharing

Data stored in a database is not usually held solely for the use of one person. A database is normally expected to be accessible by more than one person, perhaps at the same time. Hence a students' database might be accessible by members of not only academic but also administrative staff.

Data integration

Shared data brings numerous advantages to the organization. Such advantages, however, only result if the database is treated responsibly. One major responsibility of database usage is to ensure that the data is integrated. This implies that a database should be a collection of data, which, at least ideally, has no redundant data. Redundant data is unnecessarily duplicated data.

In the past, for instance separate files of student information might have been maintained by different academic and administrative departments of a university with many Fields in common. The aim of a database system would be to store one logical item of data in one place only. Hence, one student record would be accessible to a range of information systems.

Data integrity

Another responsibility arising as a consequence of shared data is that a database should display integrity. In other words, that the database should accurately reflect the universe of discourse that it is attempting to model.

Data security

One of the major ways of ensuring the integrity of a database is by restricting access. In other words, securing the database. The main way that this is done in contemporary database systems is by defining in some detail a set of authorized users of the whole, or more usually parts of the database. For instance, a secure system would be one where the finance department has access to information used for the collection of student fees but is prohibited from changing the fee levels of given students.

Data abstraction

A database can be viewed as a model of reality. The information stored in a database is usually an attempt to represent the properties of some objects in the real world.

Data independence

One immediate consequence of abstraction is the idea of buffering data from the processes that use such data. The ideal is to achieve a situation where data organization is transparent to the users or application programs which feed off data. If for instance, a change is made to some part of the underlying database no application programs using affected data should need to be changed. Also, if a change is made to some part of an application system then this should not affect the structure of the underlying data used by the application.

These properties amount to desirable features of the ideal database. As we shall see, properties such as data independence are only partly achieved in current implementations of database technology.

What is a Database Management System?

A database management system (DBMS) is an organized set of facilities for accessing and maintaining one or more databases. A DBMS is a shell which surrounds a database or series of databases and through which all interactions take place with the database. The interactions catered for by most existing DBMS fall into three main groups (see figure 1.1):

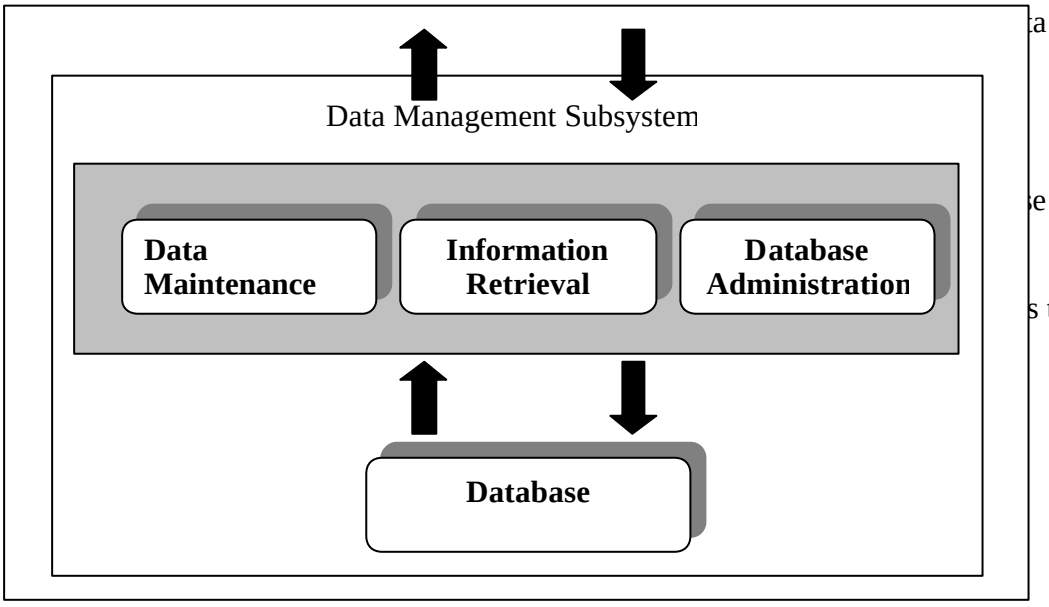


Figure 1.1 -Facilities of a DBMS

Universe Of Discourse

A database is a model of some aspect of the reality of an organization. It is conventional to call this reality a universe of discourse (UOD). Consider, for instance, an academic setting such as a university. The UOD in this case might encompass, amongst other things, modules offered to students and students taking modules.

Modules and students are examples of things of interest to us in a university. We call such things as interest classes or entities. We are particularly interested in the phenomenon that students take offered modules. This facet of the UOD would be regarded as a relationship between those classes or entities.

A class or entity such as module or student is normally defined as such because we wish to store some information about occurrences of the class. In other words, classes have properties or attributes. For instance, students have names, addresses and telephone numbers; modules have titles and credit points or current rolls.

A database of whatever form, electronic or otherwise, must be designed. The process of database design is the activity of representing classes, attributes and their relationships in a database. (This process is illustrated in figure 1.1)

Fact Bases

A Database can be considered as a well regained collection of data, which is meant to represent some UOD. Data are facts about the domain. A datum, a unit of data, is one symbol or a collection of symbols that is used to represent something that within the domain. Facts by themselves are meaningless; to prove usefulness they must be interpreted. Therefore information is interpreted row data. Information is data placed within a meaningful context. Information is data with an assigned semantic-meaning.

Consider the string of symbols 55. Taken together these symbols form a datum. Taken together however they are meaningless. To turn them into information we have to supply a meaningful context. We can interpret them this symbol as a student number, a student's age, or the number of students taking a module. Information of this sort will contribute to our knowledge of a particular domain, in this case educational administration.

A database can be considered as a collection of facts or positive assertions about a UOD, such as relational database design is a module and John Davies takes relational database design. Usually negative facts, such as what modules are not taken by a student, are not stored. Hence, databases constitute 'closed worlds' in which only what is explicitly represented is regarded as being true.

A database is said to be in a given state at a given time. A state denotes the entire collection of facts that are true at a given in time. A database system can therefore be considered as fact base, which changes with the time.

Persistence

Data in database is described as being persistent. By persistent we mean the data is held for some duration. The duration may not actually be very long. The term persistence is used to distinguish more permanent data from data, which is more transient in nature. Hence, product data, account data, patient data and student data would all normally be regarded as examples of persistent data. In contrast, data input at a terminal, held for manipulation within a program, or printed out on a report wouldn't be regarded as persistent; as once it has been used it is no longer required.

Intentional and Extensional Parts

A database's made up of two parts: an intentional part and an extensional part. The intension of a database is set of definitions, which describe the structure of a given database. The extension of a database is total set of data stored in database. The intension of a database is also referred to as its schema. The activity of developing a schema for a database system is referred to as database design.

Below, for instance, we informally define the schema relevant to a university database:

Schema: university

Classes:
- Modules - courses run by the institution in an academic semester
- Students - people taking modules at the institution

Relationships:
- Students take Modules

Attributes:
- Modules have names
- Students have names

In the schema we have identified classes of things such as modules, relationships between classes such as students take modules, and properties of classes such as modules have names. The process of developing such definitions is considered in detail in chapters 2 and 3.

A brief extension for the academic database might be:

Extension: university

Modules:
- Computer Science
- System Analysis and Design

Students:
- Anne Johon
- Peter Jones
- Miller L.H

Takes:
- Miller L.H takes System Analysis and Design

Integrity

When we say that a database displays integrity we mean that it is an accurate reflection of its UOD. The process of ensuring integrity is a major feature of modern information systems. The process of designing for integrity is a much neglected aspect of database development.

Integrity is an important issue because most databases are designed, once in use, to change. In other words, the data in a database will change over a period of time. If a database does not change, i.e. it is only used for reference purposes, then integrity is not an issue of concern.

It is useful to think of database change as occurring in discrete rather than continuous time. In this sense, we may conceive of a database as undergoing a number of state-changes, caused by external and internal events. Of the set of possible future states feasible for a database some of these states are valid and some are invalid. Each valid state forms the extension of the database at that moment in time. Integrity is the process of ensuring that a database travels through a space defined by valid states.

Integrity involves determining whether a transition to the state below is a valid one. That is, integrity involves answering questions such as: is it valid to add another data recode university database, which relates Miller L.H to Computer Science?

Extension: university

Modules:
- Computer Science
- System Analysis and Design

Students:
- Anne Johon
- Peter Jones
- Miller L.H

Takes:
- Miller L.H takes System Analysis and Design
- Miller L.H takes Computer Science

In this case it is probably would be valid transition.

Replication

When we design the database should design to minimize replication of data. In a database we attempt to store only one item of data about one object or relationships between objects in our UOD. Ideally, a database should be a repository with no replicated facts.

If we try to add the assertion Miller L.H takes Computer Science to our extension. This is not a valid transition. As we shall see, clearly adding this assertion to our extension will replicate the relationship between Miller L.H and Computer Science

Transaction

The events that cause a change of state of the database is characterized in database terms as transaction. A transaction changes a database from one state to another.

A transaction type that might be relevant to the university database system might be:

Enroll Student in Module

Integrity Constraints

Database integrity is ensured through integrity constraints. An integrity constraint is a rule, which establishes how a database is to remain an accurate reflection of its UOD.

Constraints may be divided into two major types: static constraints and transition constraints.

A static constraint or “state invariant” is used to check that an incoming transaction will not change a database into an invalid state. A static constraint is a restriction defined on states of the database. An example of a static constraint relevant to our University database might be: students can only take currently offered modules only. This static constraint would prevent us from entering the following fact into our current database:

Miller L.H takes Deductive Database Systems

Since Deductive Database Systems is not a currently offered module by the university.

In contrast, a transition constraint is a rule that relates given states of a database. A transition is a state transformation and can therefore be denoted by a pair of states. A transition constraint is a restriction defined on a transition. An example of a transition constraint might be: the number of modules taken by a student must not drop to zero during a semester. Hence, if we wished to remove a fact relating Miller to a particular module from our database it would first check that this would not cause an invalid transition.

Database Functions

Most data held in a database is there to fulfill some organizational need. To perform useful activity with a database we need two types of function: Update and Query functions. Update functions cause changes to data. Query functions will extract data from the database.

Update Functions

A transaction is an update function. It changes a database from one state to another. Update functions that might be relevant to the university database might be:

Update Functions:
- Initiate Semester
- Offer Module
- Cancel Module
- Enroll student on course
- Enroll student in module
- Transfer Student between modules

Query Functions

The other major type of function is the query function. This does not modify database in any way, but is used primarily to check whether a fact or group of facts holds in a given database. As we shall see, query functions can use to retrieve the data from the database.

- Is course X being offered?
- Is Student Y takes Course X?

Above Query functions will relevant to our OUD.

Formalisms

Every database system must use some representation formalism. Patrick Henry Winston has defined a representation formalism as being, “a set of syntactic and semantic conventions that make it possible to describe things” (Winston, 1984). The syntax of a representation specifies a set of rules for combining symbols and arrangements of symbols to form statements in the representation formalism. The semantics of a representation specify how such statements should be interpreted. That is, how can we derive meaning from them?

In database terms the idea of representation formalism corresponds with the concept of a data model. A data model provides for database developers a set of principles by which they can construct a database system.

Multi-User Access

A database can be used solely by one person or one application system. In terms of organizations however, many databases are used by multiple users. In such situations we speak of a multi-user database system, By definition some way must exist in a multi-user database of handling situations where a number of persons or application systems want to access the same data at effectively the same time.

Consider a situation where one user is enrolling student Anne John on module Relational Database Systems. At the same time another user is removing the module Relational Database Systems from the current offering. Clearly, in the time it takes the first user to enter an enrolment fact, the second user could have denied the module fact. The database is left in an inconsistent state.

In any multi user database system some system must therefore be provided to resolve such conflicts of concurrency.

Database Views

Part of the reason that data in databases is shared is that a database may be used for different purposes within one organization. For instance, the academic database described in this chapter might be used for various purposes in a university such as recording student grades or timetabling classes. Each distinct user group may demand a particular subset of the database in terms of the data it needs to perform its work. Hence administrators in academic departments will be interested in items such as student names and grades while a timetabler will be interested in rooms and times. This subset of data is known as a view.

In practice a view is merely a query function that is packaged for use by a particular user group or program. It provides a particular window into a database and is discussed in detail in chapter 10.

Evolution of the Databases

Database systems are tools for data management and form a branch of information technology. Data management is an ancient activity and the use of information technology for such activity is a very recent phenomenon. Gray (1996) provides a useful categorization of data management into six historical phases:

Phase 01: Manual record managers (4000 BC-1900 AD)

Human beings have been keeping data records for many thousands of years. For example, the first known writing describes the royal assets and taxes in Sumeria dating from around 4000 BC. Although improvements were made in the recording medium with the introduction of paper, little radical change was made to this form of manual data management for 6000 years.

Phase 02: Punched-card record managers (1900-1955)

The invention of automated data management began with the invention of the Jacquard loom circa 1800. This technology produced fabric from patterns represented on punched cards. In 1890, Herman Hollerith applied the idea of punched cards to the problem of producing the US census data. Hollerith formed a company (later to become International Business Machines - IBM) to produce equipment that recorded data on such cards and was able to sort and tabulate the cards to perform the census. Use of punched-card equipment and electromechanical machines for data management continued up until the mid-1950s.

Phase 03: Programmed record managers (1955-1970)

Stored program computers were developed during the 1940s and early 1950s to perform scientific and military calculations. By 1950 the invention of magnetic tape caused a significant improvement in the storage of data. A magnetic tape of the time could store the data equivalent to 10,000 punched cards. Also, the new stored-program computers could process many hundreds of records per second using batch processing of sequential files. The software of the day used a file-oriented record-processing model for data. Programs read data as records sequentially from several input files and produced new files as output. A development of this approach was batch transaction processing systems. Such systems captured trans-actions on card or tape and collected them together in a batch for later processing. Typically these batches were sorted once per day and merged with a much larger data-set known as the master File to produce a new master file. This master file also produced a report that was used as a ledger for the next day's business.

Batch processing used the computers of the day very efficiently. However such systems suffered from two major shortcomings. First, errors were not detected until the daily run against the master file, and then they might not be corrected for several days. Second, the business had no way of accurately knowing the current state of its data because changes were only recorded in the batch ran, which typically ran overnight.

Phase 04: On-Line Network Data managers (1965-1980)

Applications like stock-market trading and travel reservation could not use the data supplied by batch processing systems because it was typically at least one day old. Such applications needed access to current data. To meet such a need many organizations began experimenting with the provision of on-line transaction databases, such databases became feasible with developments in teleprocessing monitors, which enabled multiple terminals to be connected to the same central processing unit (CPU). Another key development was the invention of random access storage devices such as magnetic drums and disks. This, in turn, led to improvements in file structures with the invention of indexed sequential files. This enabled data management programs to read a few records off devices, update them, and return the updated records for inspection by users.

The indexed-sequential file organization led to the development of a more powerful set-oriented record model. This structure enabled applications to relate two or more records together in hierarchies. This hierarchical data model evolved into a more flexible representation known as the network data model.

Managing set-oriented processing of data became so commonplace that the Cobol programming language community chartered a database task group (DBTG) to define a standard data definition and manipulation language for this area. Charles Bachman, who had built a prototype data navigation system at General Electric (called the Integrated Data Store - IDS) received a Turing award for leading the DBTG effort. In his Turing lecture Bachman described this new model of data management in which programs could navigate among records by traversing the relationships between them.

The DBTG model crystallised the concept of schemas and data independence through the definition of a three-level architecture for database systems (chapter 1). These early on-line systems pioneered solutions to running many concurrent update activities against a database shared amongst many users. This model also pioneered the concept of transactions and the management of such transactions using locking mechanisms and transaction logs.

A person named Goodrich saw the work that was being done at GEC and decided to port IDS across onto the new IBM system 360 range of computers. John Cullinane entered into a marketing agreement with Goodrich. This was the begin-ning of a company named Cullinane, later CuUinet, which established the IDMS DBMS as the dominant force of network DBMS on IBM mainframes in the 1960s, 1970s and 1980s.

Phase 05: Relational Data Management (1980-1995)

Despite the success of network databases, many software practitioners felt that navigational access to data was too low-level and primitive for effective application building. In 1970, E.F. Codd outlined the relational data model which offered higher-level definition and manipulation of data.

In 1970 an IBM scientist Dr. E.F. Codd published an influential paper on database architecture entitled 'A Relational Model for Large Shared Data Banks' (Codd 1970). Researchers at IBM used the material in Codd's early publications to build the first prototype relational DBMS called System/R. This was emulated at a number of academic institutions, perhaps the foremost example being the INGRES research team at the University of Berkeley, California.

During the 1970s and early 1980s relational databases got their primary support from academic establishments. The commercial arena was still dominated by IDMS-type databases. In 1983, however, IBM announced its first relational database for large mainframes - DB2. Since that time, relational databases have grown from strength to strength.

Many people claim that relational systems got their strongest impetus from the large number of PC-based DBMS created during the 1980s. Although it is undoubtedly true that many of these packages such as dBase II and its derivatives were relational-like, there is some disagreement over whether they truly represented relational DBMS. The most popular current example of this class of DBMS is Microsoft Access

The relational model was influential in stimulating a number of important developments in database technology. For example, developments in client-server com-putting, parallel processing and graphical user interfaces were built on the bedrock supplied by the relational data model.

The idea of applying parallel computer architectures to the problems of database management have something of the order of twenty years of history. Only comparatively recently however has the idea been treated seriously in the development of commercial database systems.

Phase 06: Multimedia Databases (1995-)

Databases are now being used for storing richer data types - documents, images, voice and video data, as storage engines for the Internet and intranet, and offer the ability to merge procedures and data. Traditionally, a database has stored simply data. The processing of data was accomplished by application programs working outside of, but in cooperation with, a DBMS. One facet of many modern relational DBMS is their ability to embed processing within the database itself. This has a number of advantages, particularly in its suitability for client-server architectures.

In recent years many claims have been made for new types of database systems based on object-oriented ideas. A number of commercial object-oriented DBMS have now become available. Many of the relational vendors are also beginning to offer object-oriented features in their products particularly to enable handling complex data.

Database Architecture

The ANSI/SPARC Model

Modern database technology require sound framework for database system architecture. To fulfill this requirement for databases the ANSI/SPARC Study Group defined a three –level architecture for typical database systems:

The internal level is the one closest to physical storage - i.e. the one concerned with the way the data is actually stored.

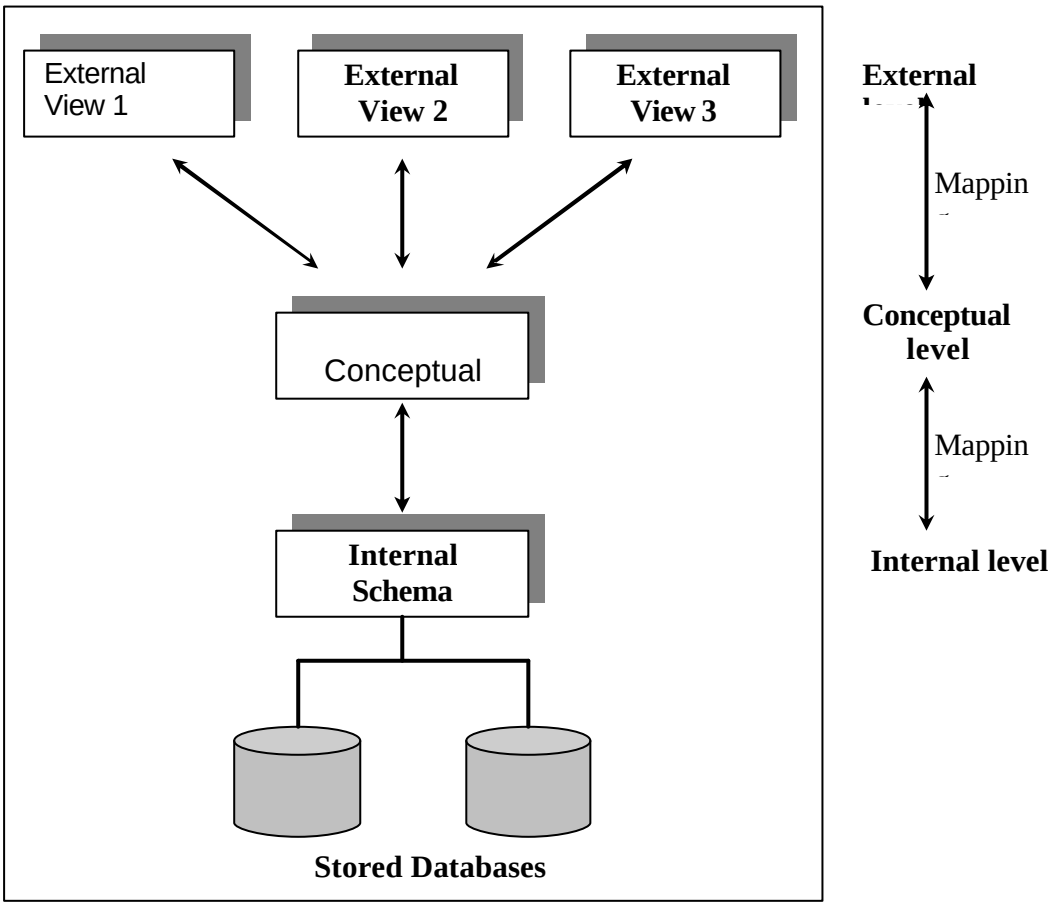


Figure 1.2

The ***internal level*** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

The ***external level*** is the one closest to the users - i.e. it is the one concerned with the way the data is viewed by individual users. The external level includes a number of external schemas or user views. Each external schema describes the database view of one group of database users. Each view typically describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high level data model or an implementation data model can be used at this level.

The ***conceptual*** level is a level of indirection between the two. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema is a global description of the database that hides the details of physical storage structures and concentrates on describing entities, data types, relationships, and constraints. A high-level data model or an implementation data model can be used at this level.

Mappings

In a DBMS based on the three schema architecture, each user group refers only to its own external schema which is converted into a request on the conceptual schema, then into a request on the internal schema for processing on the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view before it is presented to the user.

The processes of transforming requests and results between levels are called mappings. There are two levels of mapping in the architecture, one between the external and conceptual levels of the system and one between the conceptual and internal levels.

The ***conceptual/internal mapping*** defines the correspondence between the conceptual view and the stored database; it specifies how conceptual records and fields are represented at the internal level. If the structure of the stored database is changed i.e. if a change is made to the storage structure definition - then the conceptual/internal mapping must also be changed accordingly, so that the conceptual schema may remain invariant.

An ***external/conceptual mapping*** defines the correspondence between particular external view and the conceptual view. The differences that may exist between these two levels are similar to those that may exist between the conceptual view and the stored database. For example, fields can have different data types, field and record names can be changed, and multiple conceptual fields can be combined into a single (virtual) external field, and so on. Any number of external views can exist at the same time; any number of users can share a given external view; different external views can overlap.

Data Models

Every database, and indeed every DBMS, must adhere to the principles of some data model. However, the term data model is somewhat ambiguous. In the database system literature the term is used in a number of different senses, two of which are the most important: that of architecture for data; that of an integrated set of data requirements.

Data Model as Architecture

In this sense of the term, data model is used to refer to a set of general principles for handling data. Here, people talk of the relational data model, the hierarchical data model or Object oriented data model.

The set of principles that defines a data model may be divided into three major parts:

1. Data definition - A set of principles concerned with how data is structured.
2. Data manipulation - A set of principles concerned with how data is operated on.
3. Data integrity - A set of principles concerned with determining which states are valid for a database.

Data definition involves defining an organization for data: a set of templates into which data will be fitted.

Data manipulation concerns the process of how the data is accessed and how it is changed in the database. Data integrity is very much linked with the idea of data manipulation in the sense that integrity concerns the idea of what are valid changes and invalid changes to data.

Any database and DBMS must adhere to the tenets of some data model. Hence, in the relational data model, data definition involves the concept of a relation, data manipulation involves a series of relational operators, and data integrity amounts to two rules - entity and referential integrity. Note that by the data integrity part of a data model we are describing only those rules those are inherently part of the data model. A great deal of other integrity constraints or rules will have to be specified by additional means, i.e. using mechanisms not inherently part of the data model.

Data Model as Blueprint

In this sense, the term data model is used to refer to an integrated, but implementation independent, set of data requirements for some application. Here, analysts might speak of the order-processing data model, the accounts-receivable data model, or the student-admissions data model. A data model in this sense is an important component part of any information systems specification (Beynon-Davies, 1993). To avoid confusion, whenever it is not explicit from the context we shall prefix the first type of data model with the word architectural and the second type of data model with the word applications.

Typology of Architectural Data Models

We may make a distinction between three generations of architectural data model (Brodie etal, 1984);

1. *Primitive Data Models.*

In this approach objects are represented by record-structures grouped in file-structures. The main operations available are read and write operations over records.

2. *Classic Data Model.*

These are the hierarchical, network and relational data models. The hierarchical data model is an extension of the primitive data model discussed above. The network is an extension of the hierarchical approach. The relational data model is a fundamental departure from the hierarchical and network approaches.

3. *Semantic Data Models*

The main problem with classic data models like the relational data model is that they maintain a fundamental record-orientation. In other words, the meaning of the information in the database - its semantics – is not readily apparent from the database itself. Semantic information must be consciously applied by the user of databases using the classic approach. For this reason, a number of so-called semantic data models have been proposed. Semantic data models (SDMs) attempt to provide a more expressive means of representing the meaning of information than is available in the classic models. In many senses the object-oriented data model can be regarded as a semantic data model.

Database Schema and Instances

The database schema of a database is set of definitions, which describe the structure of a given database. The schema of a database is also referred to as its intension. The activity of developing a schema for a database system is referred to as database design. Below, for instance, we informally define the schema relevant to a university database:

Schema: university

Classes:
- Modules - courses run by the institution in an academic semester
- Students - people taking modules at the institution

Relationships:
- Students take Modules

Attributes:
- Modules have names
- Students have names

In the schema we have identified classes of things such as modules, relationships between classes such as students take modules, and properties of classes such as modules have names. The process of developing such definitions is considered in detail in chapters 2

According to the ANSI/SPRC model we can have 2 types of schema, Conceptual schema and internal schema. With the use of Database Language such as SQL2 user can developed named schemas to make the structure f the database.

Distributed Databases

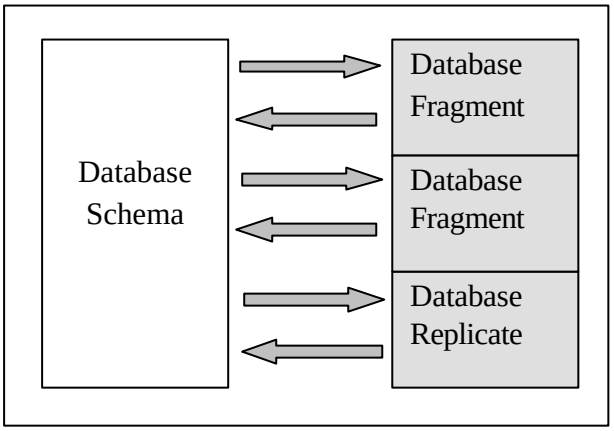


Figure 1.3 Distributing data

In contemporary database systems the extension and the intension of the database may be distributed across numerous different sites. In a distributed database system one effective schema is stored in a series of separate fragments and/or copies (replicates) (see figure 1.3).

For instance, in an academic domain data about students may be held as fragments relevant to each school or department in a university. A replicate of some or all of this data may be held for central administration purposes.

CHAPTER 2

Database Design and Modeling

Introduction

This chapter of this course is concerned with the general subject of database design and modeling (more specifically. relational database design). The database design problem can be stated very simply: Given some body of data to be represented in a database, how do we decide on a suitable logical structure for that data"—in other words, how do we decide what entities should exist and what attributes they should have? The practical significance of this problem is obvious.

Before we start getting into details, a number of preliminary remarks are in order:

1. First, note that we are concerned here with logical (or conceptual) design and physical design. Because physical design is also very important same as conceptual design. However:
2. Physical design can be treated as a separate, follow-on activity. In other words, the "right" way to do database design is to do a clean logical (i.e., relational) design first, and then, as a separate and subsequent step, to map that logical design into whatever physical structures the target DBMS happens to support. (In other words. as noted in this Chapter, the physical design should be derived from the logical design. not the other way around.)
3. Physical design, by definition, tends to be somewhat DBMS-specific. Logical design, by contrast, is or should be quite DBMS-independent, and there are some Solid theoretical principles that can be applied to the problem.

The structure of this chapter is as follows. First in this Chapter lays some theoretical groundwork on the database design and modeling. And then we try to map the design process to model enterprise data base requirements. Father we concern on logical and physical database design briefly by introducing the concepts of "entity/relationship" modeling and normalization, which use to build the logical database structure. Later of this chapter will concern the different approaches such as Hierarchical, Network and relational approaches briefly.

Database design and Development process

Design a data model is an important part of the process of database development. Database development is a process of modeling. It is a process of successive refinement through three levels of model: conceptual models, logical models and physical models.

1. A conceptual model is a model of the real world expressed in terms of data requirements.
2. A logical model is a model of the real world expressed in terms of the principles of some data model.
3. A physical model is a model of the real world expressed in terms of files and access structures such as indexes.

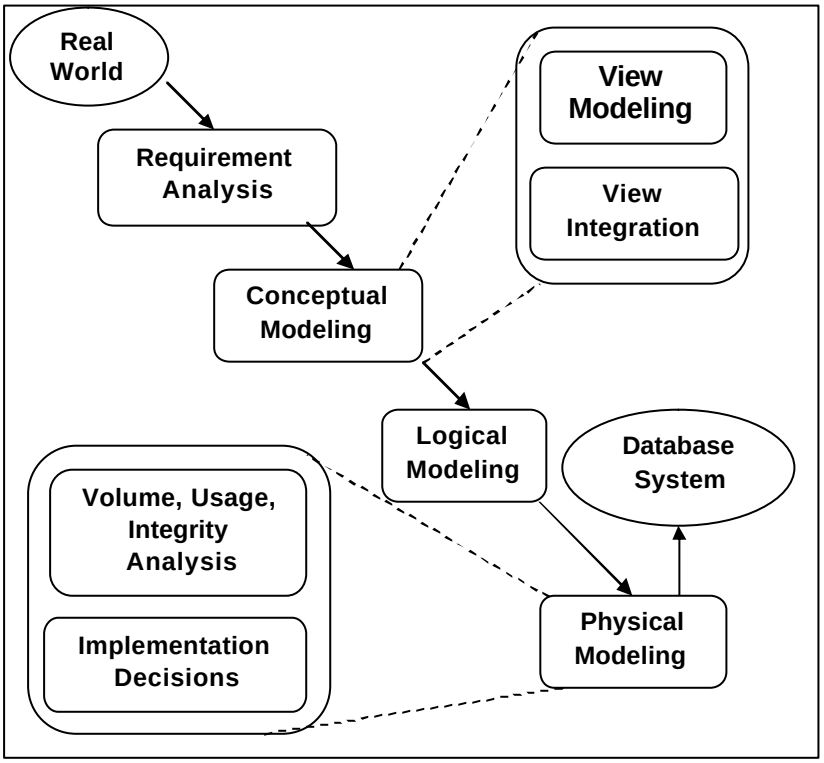


Figure 2.1: Database Development Process

There are hence three core stages to any database development task:

Conceptual modeling, logical modeling and physical modeling (see figure 2.1). Some people see conceptual modeling as a stage headed by requirements analysis stage

The process of requirements analysis involves eliciting the initial set of information and processing requirements from users.

The conceptual modeling stage can be thought of as comprising two sub stages: view modeling, which transforms the user requirements into a number of individual user views, and view integration which combines these schemas into a single global schema. Most conceptual models are built using constructs from the semantic data models. In chapter 3 we shall use the extended entity-relationship model to illustrate the process of building conceptual models.

The process of logical modeling is concerned with determining the contents of a database independently of the exigencies of a particular physical implementation. This is achieved by taking the conceptual model as input, and transforming it into the architectural data model supporting the target database management system (DBMS). This is usually the relational data model. In chapter 4 we shall outline the key concepts of the relational data model.

Physical modeling involves the transformation of the logical model into a definition of the physical model suitable for a specific software/hardware configuration. This is usually some schema expressed in the data definition language of SQL, In chapter 4 we illustrate the primary components of this modern-day database sub-language.

Data Analysis and Modeling

The term data analysis is frequently used in the context of database work. Data analysis is a term generally reserved for conceptual and logical modeling. There are two complementary approaches to conducting data analysis: normalization and entity relationship modeling.

Normalization is a technique based upon the work of Codd (Codd 1970). Sometimes referred to as a bottom-up design technique, normalization involves the transformation of data subject to a range of file maintenance problems into a form free from such problems (chapter 5).

Entity relationship modeling is sometimes known as a top-down design technique (chapter 3). Entity relationship modeling involves representing some universe of discourse in terms of entities and relationships. Object modeling is discussed in this work as an extension of entity relationship modeling. It is particularly seen as a recent attempt to use the constructs of top-down data analysis to model system behavior or dynamics

Enterprise Data Modeling

Data, Information and Knowledge

The concept of information is an extremely vague one open to many different interpretations (Stamper 1989). One conception popular in the computing literature is that information results from the processing of data: the assembly, analysis or summarization of data. This conception of information as analogous to chemical distillation is useful, but ignores the important place of human interpretation in any understanding of information.

In this section we shall define a workable definition of information based upon the distinction between data, information and knowledge. We shall then elaborate on this definition by adding people into the equation.

Tsitchizris and Lochovsky (1982) define information as being “an increment of knowledge which can be inferred from data”. Information therefore increases a person's knowledge of something. Note that this definition interrelates the concepts of data, information, knowledge and people.

1. Data is facts. A datum, a unit of data, is one or more symbols that are used to represent something.
2. Information is interpreted data. Information is data placed within a meaningful context.
3. Knowledge is derived from information by integrating information with existing knowledge.
4. Information is necessarily subjective. Information must always be set in the context of its recipient. The same data may be interpreted differently by different people depending on their existing knowledge and different contexts.

Human Activity Systems, Information Systems and Information Technology Systems

Now we try to identify the concept of an information system and its relation ship with human activity systems. We also distinguish between an information system and an information technology system. And finally walkthrough the Enterprise Data modeling to build the IS for the Organization and an information technology system.

Information System (Is)

An information system is a system which provides information for some organization or part of an organization. A system might be defined as a coherent set of interdependent components which exists for some purpose, has some stability, and can be usefully viewed as a whole. Systems are generally portrayed in terms of an input-process-output model existing within a given environment.

The environment of a system might be defined as anything outside a system which has an effect on the way the system operates. The inputs to the system are the resources it gains from its environment or other systems. The outputs from the system are those things which it supplies back to its environment or other systems. The process of the system is the activity which transforms the system inputs into system outputs. Most organizations are open systems in that they are affected by their environment and other systems.

Human Activity Systems

Information systems support human activity systems. The idea of a system has been applied in many fields as diverse as physics, biology and electronics. The class of systems to which computing is generally applied has been referred to as human activity systems (Checkland 1987). Such systems have an additional component added to the input-process-output model described above: people.

Human activity systems consist of people, conventions and artifacts designed to serve human needs. Every human activity system will have one or more information systems. The purpose of such information systems is to support and enable the effective management of the human activity system. Organizations are normally made up of a number of human activity systems and hence normally need a number of information systems to work effectively.

Information Technology (IT)

Information technology provides means of constructing aspects of modern-day information systems. Information technology (IT) includes computers (hardware and software) and communications. Computers and communication networks are primarily used to support aspects of an organization's information systems. It is important to recognize that information systems have existed in organizations prior to the invention of IT, and hence IS do not need IT to exist. However, in the modern, complex organizational world most IS rely on IT to a greater or lesser degree. Therefore within this text whenever we refer to an IS we really mean an information technology system.

The relationship between IT, IS and human activity systems is illustrated in figure 2.2.

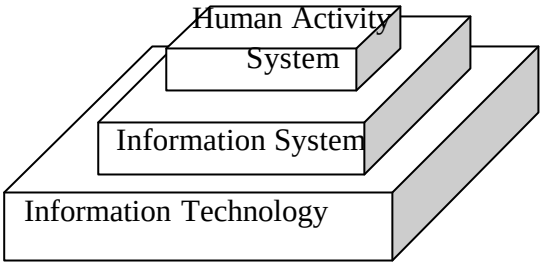


Figure 2.2: Relationship between IT, IS and Human activity system

Layers of an IT System

It is useful to consider an information technology system as being made up of a number of subsystems or layers (see figure 2.3):

1. Interface subsystem. This subsystem is responsible for managing all interactions with the end-user. Hence, it is frequently referred to as the user interface.
2. Rules subsystem. This subsystem manages the application logic in terms of a defined model of business rules.
3. Transaction subsystem. This subsystem acts as the link between the data subsystem and the rules and interface subsystems. Querying, insertion and update activity is triggered at the interface, validated by the rules subsystem and packaged as units (transactions) that will initiate actions (responses or changes) in the data subsystem.
4. Data subsystem. This subsystem is responsible for managing the underlying data needed by an application.

In the contemporary IT infrastructure each of these parts of an application may be distributed on different machines, perhaps at different sites. This means that each part usually needs to be stitched together in terms of some communications backbone. For consistency, we refer to this facility here as the communications subsystem.

Some people have also argued that a typical application is also like an ice-berg. The user interface only forms the tip of the ice-berg (some 10% of the application). The major element of the application is that part of the ice-berg that is hidden from view, the 90% 'below the water''.

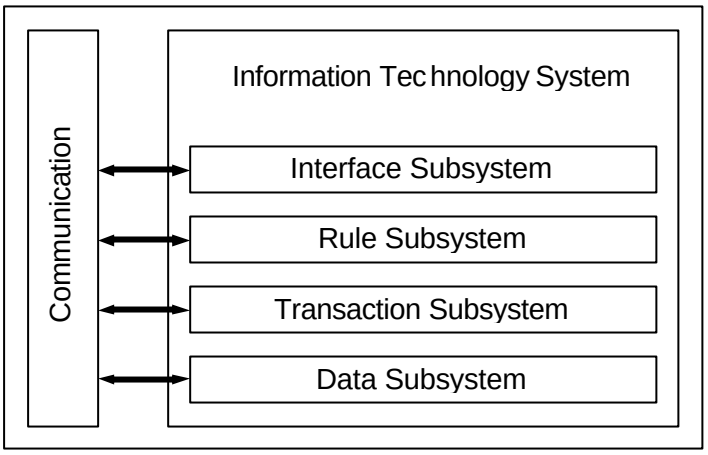


Figure 2.3: Layers of an information technology system

Historically, the four parts of a conventional IT application were constructed using one tool, the high-level or third generation programming language (3GL).

The key conclusions to be drawn from this section are that information systems support human activity systems and consequently cannot be designed without an effective understanding of the context of human activity. Information technology systems are critical components of contemporary information systems. Database systems are critical elements of information technology systems.

Application Data Models

Because of its central importance for organizations, data has to planned for and managed. The major tool employed in data planning and management is the data model.

Data models can be produced on at least three levels (see figure 2.4):

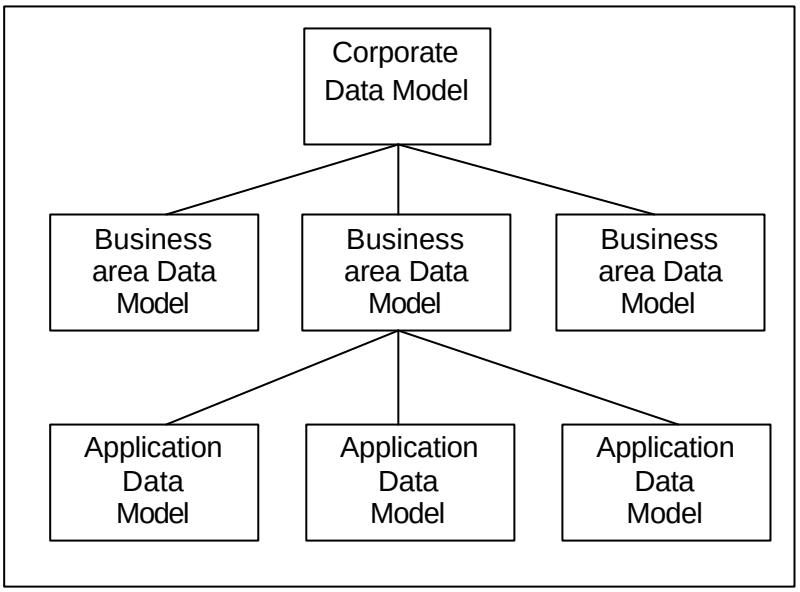


Figure 2.4: Levels of data models

1. Corporate data models. Specifying data requirements for a whole organization.

2. Business area data models. Specifying data requirements for a particular business area.

3. Application data models. Specifying data requirements for a particular information systems application.

Consider the case of a university setting. Traditionally, to support the activities of the university given application data models may have been produced to document the data requirements relevant to a particular information system such as a library information system or a student enrolment system. At a higher level a data model may be produced to integrate the data requirements relevant to a particular business area. In a university setting, for instance, we might develop a data model relevant to the key 'business' areas of teaching, research and consultancy.

At the highest level the university may have developed a corporate data model to integrate the data requirements for the entire organization. This may either be a unification of the business area data models or a summary of their key elements

Corporate Data Modeling and Is Planning

The development of application, business area and corporate data model may be critical element of IS planning. IS planning is the process of defining an information systems architecture for the organization, and developing strategic and operational plans for the delivery of this architecture. An IS architecture can be envisaged as having three inherent layers corresponding to the distinctions raised between information, information systems and information technology described above:

1. Information architecture. This consists of activities involved in the collection, storage, dissemination and use of information within the organization.

2. Information Systems Architecture. This consists of the information systems needed to support organizational activity in the areas of collection, storage, dissemination and use.

3. Information technology architecture. This consists of the hardware, software, communication facilities and IT knowledge and skills available to the organization.

There is a close relationship between the three levels of database models and the three levels of an IS architecture. This is illustrated in figure 2.5.

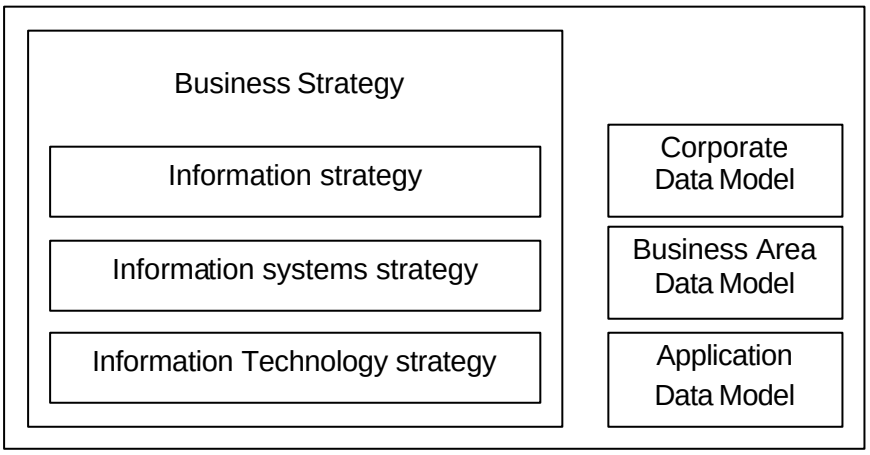


Figure 2.5

Contemporary Database Types

As we have indicated above, databases are arguably the most important contemporary component of an information technology system. In such systems databases serve three primary purposes: as production tools, as tools for supporting decision making and as tools for deploying data around the organization.

Production Databases

Such databases are used to collect operational or production data. Production databases are used to support standard organizational functions by providing reliable, timely and valid data. The primary usage of such databases includes the creating, reading, updating and deleting of data - sometimes referred to as the CRUD activities.

As an example, in terms of a university, a production database will probably be needed to maintain an ongoing record of student progression.

Decision Support Databases

Such databases are used as data repositories from which to retrieve information for the support of organizational decision-making. Such databases are read only databases. They are designed to facilitate the use of query tools or custom applications.

As an example, in terms of a university, a decision-support database may be needed to monitor recruitment and retention patterns among a student population.

Mass-Deployment Databases

These databases are used to deliver data to the desktop. Generally such databases are single-user tools running under some PC-based DBMS such as Microsoft Access. They may be updated on a regular basis either from production or decision-support databases.

As an example, in terms of a university, a mass-deployment database will be needed by each lecturer to maintain an ongoing record of student attendance at lectures and tutorials.

Ideally we would like any database system to fulfill each of these purposes at the same time. However, in practice, medium to large-scale databases can rarely fulfill all these purposes without sacrificing something in terms of either retrieval or update performance. Many organizations therefore choose to design separate databases to fulfill production, decision-support and mass deployment needs and build necessary update strategies between each type.

Developments in Database Applications

Besides their use as critical elements in conventional IT systems, databases and database systems are critical components of a number of developing applications within organizations.

Data Warehouses/Data Marts

A data warehouse is a type of contemporary database system designed to fulfill decision-support needs. A data warehouse differs from a conventional decision support database in a number of ways.

a data warehouse is likely to hold far more data than a decision-support database. Volumes of the order of 400 Giga-bytes to Tera-bytes of data are commonplace. Second, the data stored in a warehouse is likely to have been extracted from a diverse range of application systems, only some of which may be database systems. Third, a warehouse is designed to fulfill a number of distinct ways (dimensions) in which users may wish to retrieve data.

A data mart is a small data warehouse. Whereas a data warehouse may store in the order of 400 Gb of data, a data mart may store something in the order of 40 Gb of data. A data mart is also likely to store data representing a particular business area rather than representing data applicable to the entire organization.

On-Line Analytical Processing and Data Mining

Two application areas are normally discussed in association with data warehousing and data marts.

Online Analytical Processing (OLAP) comprises the dynamic synthesis, analysis and consolidation of large volumes of multi-dimensional data. Data mining comprises the process of extracting hidden patterns from large databases and using it to make decisions critical to some organization

.

Network Database Applications

Database systems have been impacted upon by developments in Internet technology through so-called Web-enabled or network database applications. Users access such applications through a Web browser on their desktop system. Browser software access and display Web pages sited on a Web server identified by a universal resource locator (URL). A Web page is basically a document a text file with HTML (HyperText Markup Language) codes inserted. HTML commands instruct the browser how to display the specified text file.

A user requests a service by specifying the URL. The specified Web page is then transferred across the network to the desktop, allowing the browser to display the page.

Traditional Web pages are static entities in the sense that links between textual information is hard-coded into the text. There are hence numerous problems experienced in maintaining many inter-connected Web pages.

For this reason there are many modem developments in dynamic Web pages. In a dynamic Web page, data on the Web page is dynamically updated by a database. This makes for the easier development and maintenance of larger scale Web applications.

Universal Servers

Traditional database applications support structured data such as numbers or character strings. Newer applications such as network database applications demand the ability to store and manipulate more complex data types such as image data, audio data and video data.

Many contemporary DBMS are hence casting themselves as so-called universal servers. The universal server approach involves the extension of DBMS to support both traditional and non-traditional data. Non-traditional data is generally supported through user-defined data types (UDTs) and user-defined functions (UDFs). UDTs also known as abstract data types define non-standard data structures. UDFs permit the users of a database to alter and manipulate UDTs.

Logical and Physical Database Design

In this part we cover a number of techniques relevant to the process of developing database systems. Techniques such as the ones discussed here are usually packaged together in some methodology for development. Figure P6.1 illustrates a simplified methodology for database development abstracted from a range of actual development methodologies for database systems.

The techniques used in the methodology naturally divide into three categories: those concerned with conceptual modeling, those concerned with logical modeling and those concerned with physical modeling. Figure 2.6 also includes one further stage in a methodology for database development: requirements analysis/elicitation.

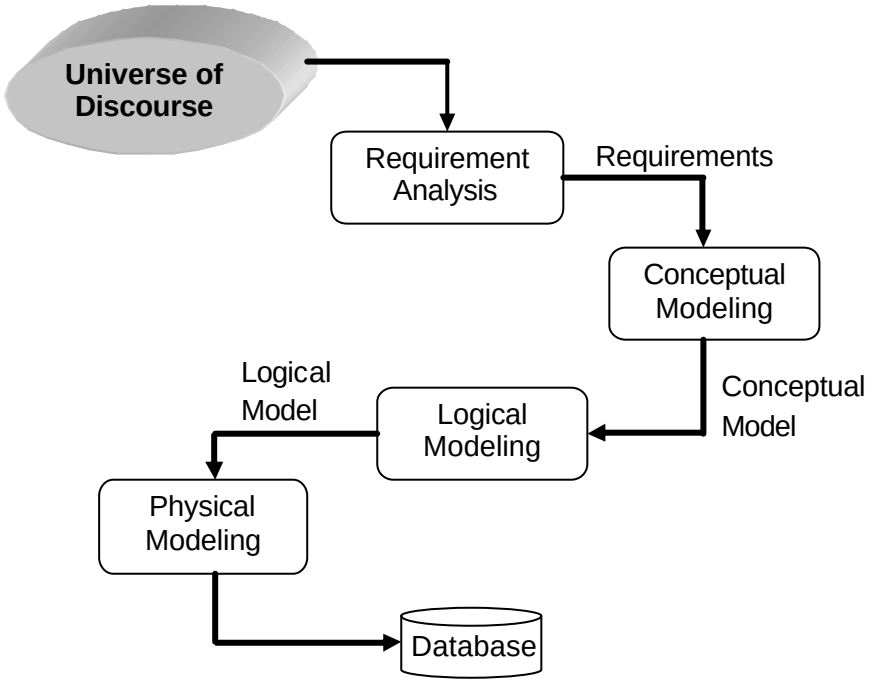


Figure 2.6: A database development methodology

Requirements analysis involves establishing the key technical requirements for a database system usually through formal and informal interaction between developers and users. In general terms it involves establishing the structure of data needed and the use of the data in some information systems context. One key aspect of requirements elicitation is the determination of the scope of the universe of discourse (UOD) to be covered by a proposed database system.

Conceptual modeling involves building a model of the real world expressed in terms of the data requirements established. There are a number of alternative approaches to conceptual modeling.

We discuss the well-established technique of entity relationship (E-R) diagramming (Chapter 5). This technique can be used to construct an entity model - a representation of a UOD in terms of entities, relationships and attributes.

Some situations we have to design particularly large databases. That time conceptual modeling may consist of two sub processes: *view modeling* and *view integration*. View modeling involves developing an application data model for a particular business area. View integration involves taking a number of these distinct views and producing an integrated data model for the organization.

Logical modeling involves constructing a model of the real world expressed in terms of the principles of some data model. Because of its popularity we focus on the relational data model in our discussion of logical modeling. In chapter 5 we will discuss how either an entity or object model may be mapped to a relational schema. In chapter 5 we consider the logical database design technique of normalization. This technique enables us to construct a relational schema free from update problems.

Physical modeling involves constructing a model of the real world expressed in terms of data structures and access mechanisms available in a chosen DBMS. This involves two distinct sub-processes: physical database design and database implementation. Physical database design comprises the process of annotating a logical model with information pertaining to a particular implementation such as volume and usage information.

Following is the process of database design, development and Implement

1. Requirements analysis
2. Conceptual modeling
   - View modeling
     - Defining entities/objects
     - Defining relationships and constraints on relationships
     - Defining attributes
     - Defining abstraction mechanisms
     - Defining behaviour
   - View integration
     - Identifying communalities among views
     - Producing a global conceptual model
     - Accommodating the conceptual model to a relational schema
3. Logical modeling
   - Normalization
     - Producing a 3NF schema through non-loss decomposition Producing a 3NF schema through a dependency analysis
     - Reconciling the normalized schema with the schema produced from conceptual modeling
4. Physical modeling
   - Physical database design
     - Volume analysis
     - Usage/transaction analysis
     - Integrity analysis
     - Control/security analysis
     - Distribution analysis
   - Database Implementation
     - Selecting the DBMS
     - Creating the physical schema
     - Establishing storage structures and associated access mechanisms
     - Adding indexes
     - De-normalization
     - Defining users and privileges
     - Tuning in terms of the chosen DBMS

- Building integrity constraints

Logical database design process

Logical database design is the process of constructing a business data model, that is, a model of the business data requirements applying to some organization or part of.

The first element of this figure indicates that logical database design will normally involve both eliciting and documenting new requirements for a database application and incorporating a range of existing data requirements perhaps gleaned from existing IT applications.

The second element of the figure indicates that two main approaches can be taken to logical database design. In a convent database project the data analysts will conduct a great deal of data modeling critical aspects of an application the analysts will also conduct some normalization. The results of these two approaches may have to be reconciled. The output of reconciliation process will be a logical schema.

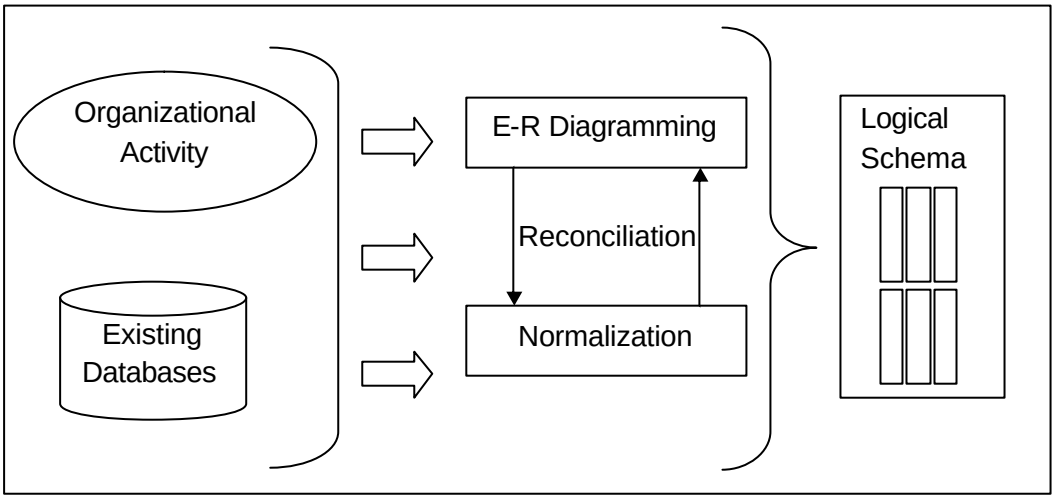


Figure 2.7: Logical database design process

Physical Database Design Process

A logical database model provides insufficient information to enable effective physical database design. Usually, the following activities need to be performed as part of physical design:

1. Volume analysis
2. Usage/transaction analysis
3. Integrity analysis
4. Control/ security analysis
5. Distribution analysis

The output from the physical database design process is an implementation plan for the database. This process is illustrated in figure 2.8. This plan will be composed of the following elements:

1. Data structures declared in a suitable DDL
2. Indexes declared on the data structures
3. Clustering data where appropriate
4. A set of inherent integrity constraints expressed in some DDL and a set of additional integrity constraints expressed in some DIL
5. A distribution strategy for the database system
6. A set of queries optimized for running on some database
7. A set of defined users
8. A plan for securing data

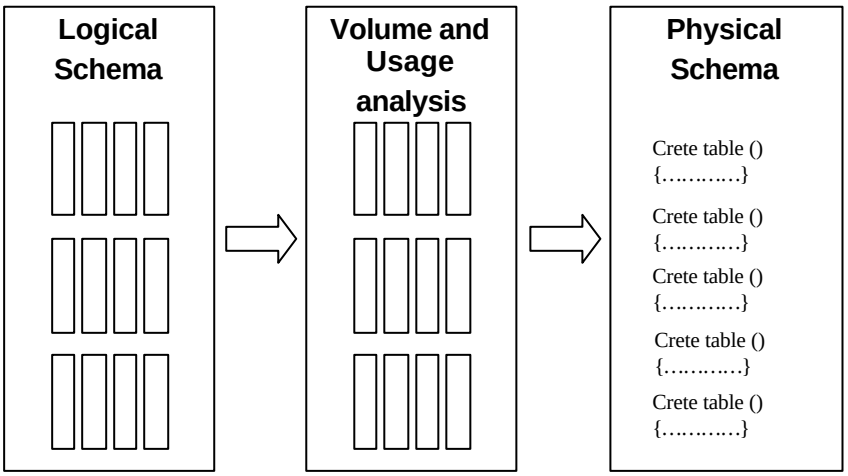


Figure 2.8: physical database design process

Volume Analysis

One of the first steps we need to take in moving from logical to physical database design is to establish estimates as to the average and maximum number of instances per entity in our database. An estimate of the maximum possible number of instances per entity is useful in deciding upon realistic storage requirements. An estimate of how many instances are likely to be present in the system on average also gives us a picture of the model's ability to fulfill access requirements.

The table below summarizes some provisional sizing estimates for the USC database. Using the column sizes established in the schema above it is relatively straightforward to translate entity sizing into table sizing. For each relation describe in the ERD.

Table	**Rows**	**Rows**	**Column**	**Table**	**Table**
	Max	***Avg***	***Size***	***Max***	***Avg***
Course	100	80	37	3,700	2,960
Lecturers	50	40	107	5,350	4,280
Students	1,000,000	800,000	83	83,000,000	24,900,000
Presentations	100,000	80,000	52	5,200,000	2,080,000
Qualifications	100	80	6	600	480
Attendance	5,000,000	1,000,000	13	65,000,000	13,000,000
Total				103,206,320	39,878,720

Usage Analysis

Usage analysis requires that we identify the major transactions required for a database system. Transactions are considered here as being made up of a series of insertions, updates, retrievals or deletions, or a mixture of all four. In University student database

- Register a new student on a presentation.
- Add new presentation details.
- Purge all lapsed presentations before a certain date.
- Assign a lecturer to a presentation.

- Record details of a possible student.

Can be sample retrieval, Update and Deletion transactions. Based on the frequency of the transactions over table of the database categorized the tables on to two groups, known as volatile (table change frequently) and Non volatile (table change infrequently)

Tables	Volatility
Courses	Low
Lecturer	Low
Attendance	High
Students	High

Transaction Analysis

Transaction analysis involves analyzing and documenting the set of critical transactions that are expected to impact against some database system. Ideally transaction analysis will fall out of conventional information systems analysis and design in the sense that these activities should document the major data management activities expected in an application.

Integrity Analysis

Proper data analysis provides a logical database design which indicates appropriate data structures and a set of inherent integrity constraints. For example, three types of inherent constraints should be documented in a relational schema

ENTITY INTEGRITY CONSTRAINTS

lecturerCode is the primary key of Lecturers and hence that lecturerCode must be unique and cannot be null.

REFERENTIAL INTEGRITY CONSTRAINTS

Every presentation must have an associated Courses record identified by courseNo. In this business, courseNo in Presentations cannot be null. Other examples are: Do not assign a non existent instructor to a session; Do not delete a course until all appropriate presentations are deleted; Do not delete an instructor until all appropriate presentations are deleted.

DOMAIN CONSTRAINTS

Domain constraints such as:

1. That courseNo in courses should be number (3) and taken from the set {<course numbers >}

2. That values for the column site in Presentations must be taken from the set {university, hotel, on-site}

Additional constraints are usually needed by any application. An important class of additional constraint is the so-called transition constraint. Such constraints document valid movements from one state of the database to another.

Remove a lecturer from qualifications if he has not given at least three presentations of a course in any one year. Static constraints involve checking that a transaction will not move the database to an invalid state. Examples of static constraints of relevance to the USC database might be:

1. Lecturer assigned to a presentation must be qualified to teach the course.
2. The duration of a presentation should equal the duration of a course.
3. The number of students booked for a presentation < = course limit.

Any constraint such as enforcing referential integrity is expensive in update performance terms. Every time a new attendance record is inserted, for instance, a check will need to be made against the Students file and the Presentations file. Constraints such as cascading updates are even more expensive in update performance terms.

Security/Control Analysis

Security is a large issue involving securing buildings and rooms, hardware, operating systems and DBMS. Many organizations now employ special persons to deal with information security issues. In this section we concentrate on the issue of database security. However, there is little point in securing a DBMS and database by itself. To take an example, most DBMS run on top of native operating systems and place their database tables within operating system files. Hence one must secure the operating system against attack by unwanted persons as well as securing the tables within the database.

Database security is normally assured by using the data control mechanisms available under a particular DBMS. Data control comes in two parts: preventing unauthorized access to data , preventing unauthorized access to the facilities of a particular DBMS. Database security will be a task for the DBA (chapter 10) normally conducted in collaboration with the organization's security expert.

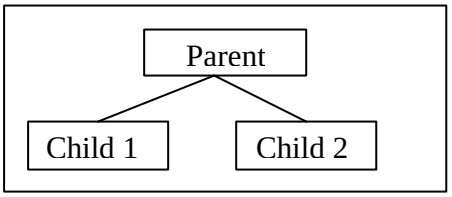
Distribution Analysis

The ability to distribute data among different nodes in a network is now a commonplace feature of modern DBMS. The design for distributed database systems is therefore an important aspect of modern database design. Distributed database design can be seen as a variant of physical database design. This we can discuss chapter 11 in detail.

Application of Database Models to database design

Hierarchical approach

The hierarchical data model does not have the strong theoretical foundation like relational data model. Probably the most prominent of DBMS adhering to a hierarchical approach is IBM's IMS (Information Management System) (see History of the databases). in this section we provide a brief description of a number of key features which characterize all hierarchical data model.



The hierarchical data model uses two data structures: record types and parent child relationships. A record type is a named data structure composed of a collection of named fields such as courseCode and courseName. Each field is used to store a simple attribute and is given a data type such as integer, character, etc. A parent child relationship is a one-to-many relationship between two record types. The record type at the one end of a relationship is said to be the parent record type, such as course; that at the many end the child record type, such as module. Hence, a hierarchical schema is made up of a number of record types related by parent child relationship.

Consider, for instance, the relationship between courses, modules and students. Courses can be considered the parent record type of modules in the sense that there are many modules making up one course. The record-type modules can in turn be considered the parent record type of students as there are many students

Based on the above relation we can develop following schema as below:

SCHEMA: University

RECORD: Course

PARENT: none

FIELDS (

courseCode: CHARCTER (6)
courseName: CHARCTER (10)
validationYear: DATE))

KEY: courseCode

ORDER BY courseCode

RECORD: Modules

PARENT: Course

FIELDS (

moduleName: CHARCTER (20)
staffnNo: INTEGER(6)
level: INTEGER(1))

KEY: moduleName

ORDER BY moduleName

RECORD: Student

PARENT: Module

FIELDS (

studentNo: INTEGER(6)

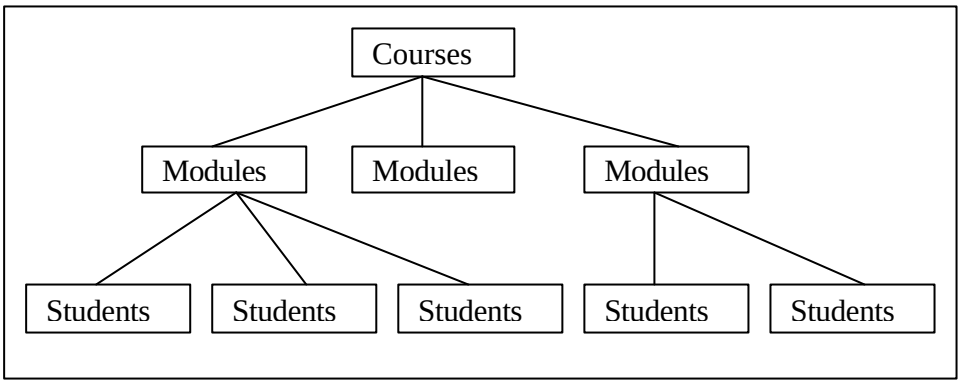
studentName: CHARCTER(20)

termAddress: CHARCTER (30))

KEY: studentNo

ORDER BY studentName

We can represent above schema in pictorial way as follows.



Note that this schema has many similarities with the relational schema describe in next section. The key differences are:

1. That the data structures are different. In the hierarchical data model we have the record type, while in the relational data model we have the relation.

2. Relationships are implemented differently. In the hierarchical data model relationships are implemented different via parent-child links. In relational model relationships are implemented via foreign key.

In the hierarchical data model, data manipulation is accomplished by embedding database access functions within some standard programming language (a host language).

There are a number of inherent integrity constraints in the hierarchical model. Two examples of such constraints are given below:

1. No record occurrence, except a root record, can exist without being linked to a parent-record occurrence. This means that a child record cannot be inserted unless it is linked to a parent record and also that deletion of a parent record causes automatic deletion of all linked child records.
2. If a child record type has two or more parent record types, then a child record must be duplicated once for each parent record.

Network Approach

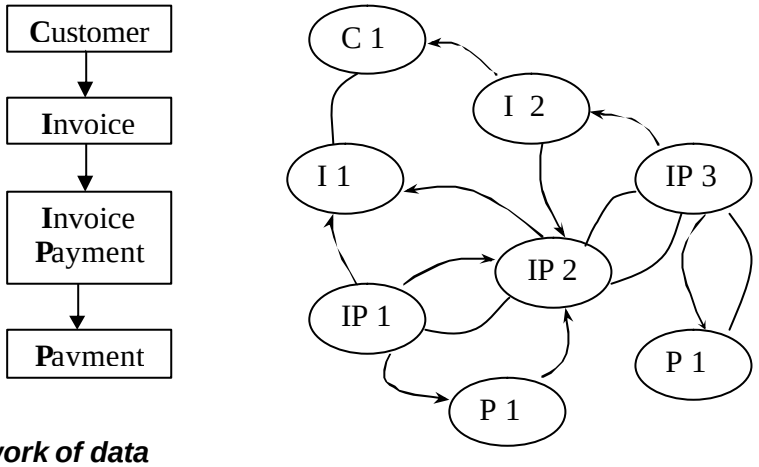
The network data model is the successor to the hierarchical data model. The dominant influence in the development of the network data model was a series of proposals put forward by the Database Task Group (DBTG) of the Conference on Data Systems and Languages (CODASYL) in 1971 (DBTG 1971). During the 1970s the majority of commercial DBMS adhered, albeit loosely, to a network data model. More recently, ANSI made a recommendation for a network definition language (NDL) in 1986.

The network model represents a complex structure. Within a network any record may have many immediate parents as well as many dependants.

Like the hierarchical model, the network data model has two data structures: *record types and set types* .

A record type is similar in concept to the record type of the hierarchical model except that fields may be used to store multiple values or represent a composite of values which repeat. For example, a record type students may have the following fields: studentNo, studentName and studentProfile. StudentProfile could be considered a composite in that it clusters together a repeating group made up of courseName, year and grade.

A set type is a description of a one-to-many relationship between two record types.



Network of data

In the Network data model, data manipulation is accomplished by embedding database access functions within some standard programming language as hierarchical model (a host language).

Relational Approach

In relational approach database is construct by s et of relations. A relation is a table, which obeys set of restrictions. Tables should conforms to all of these rules and hence constitute a relations.

Module			
Module Name	**Level**	**Course code**	**Staff No**
Relational Data Base Design	1	CSD	244
Relational Data Base Systems	1	CSD	244
Deductive Data bases	4	CSD	245
Object Oriented Databases	4	CSD	246

Lecturers		
Staff No	**Staff Name**	**Status**
244	David T	L
245	Johon P	SL
246	Evan R	PL

Courses	
Courses Code	**Module Name**
CSD	Relational Data Base Design
	Relational Data Base Systems
	Deductive Data base
	Object Oriented Database
BSD	Intro to business
	Basic Accountancy

Entire database is consisting of set of tables as above. Columns of tables are known as attributes. Rows of tables are known as tuples. The number of columns in a table is referred to as the table's degree. The number of rows in a table is referred to as the table's cardinality. Hence the cardinality of the Modules relation above is 4 and the degree of the relation is 4.

Each relation must have a primary key. This will eliminate duplication of records in table (data redundancy). In lecturer table staffNo is the primary key and this key eliminate duplication of same staff record. And each table must have foreign key, this key will make relationship between the other related tables. In relational model data retrieval is conduct by the using Set of operations known as *Relational Algebra*

(this topic we will discuss in chapter 4)

CHAPTER 3

Data Modeling Using ER Model

Introduction

An entity is a 'thing' about which an organization holds information. Entity modeling is a technique for showing relationships between entities. Entity analysis is the process by which the model is developed, and identifies the underlying structure of the data and relationships of those data. The diagram produced from entity modeling is called the Entity Relationship Diagram (ERD). Computer systems, which store large quantities of data, need to retrieve the data rapidly, in a variety of sequences and combinations. Problems can often result, whether the system uses a database or individual files, because of the way in which individual data items are stored within the computer system; this may not reflect the underlying structure of the data that exists in the business world outside the computer system

ER Model Concepts

Entity analysis is performed at two stages during the analysis phase of a system:

During analysis of the existing system, to aid the analyst's understanding of it;

In conjunction with relational data analysis, (see Chapter 5) to produce a model of the required data structure for the new system.

Ideally, analysts beginning work on a new project do not have to start the ERD from a blank sheet of paper. Many organizations have recognized the importance of their data sufficiently to develop a corporate data model, which is just a company-wide ERD.

Such a model ensures that the underlying structure of data from the point of view of the organization as a whole is appreciated. If a corporate ERD is available, the individual project's ERD will just be a subset of this, and the analysts, whilst developing more detail in relation to the entities within their particular project area, will have a framework to guide them. However even if such a corporate data model does exist, entity analysis is still essential to the individual project.

For example, an order processing system has following data model.

Suppler	who supply products
Item	the different types of items
Product	the different types of products
Customer	who purchase the products
Order	Order details of customers
Shipment	Shipment details of Items

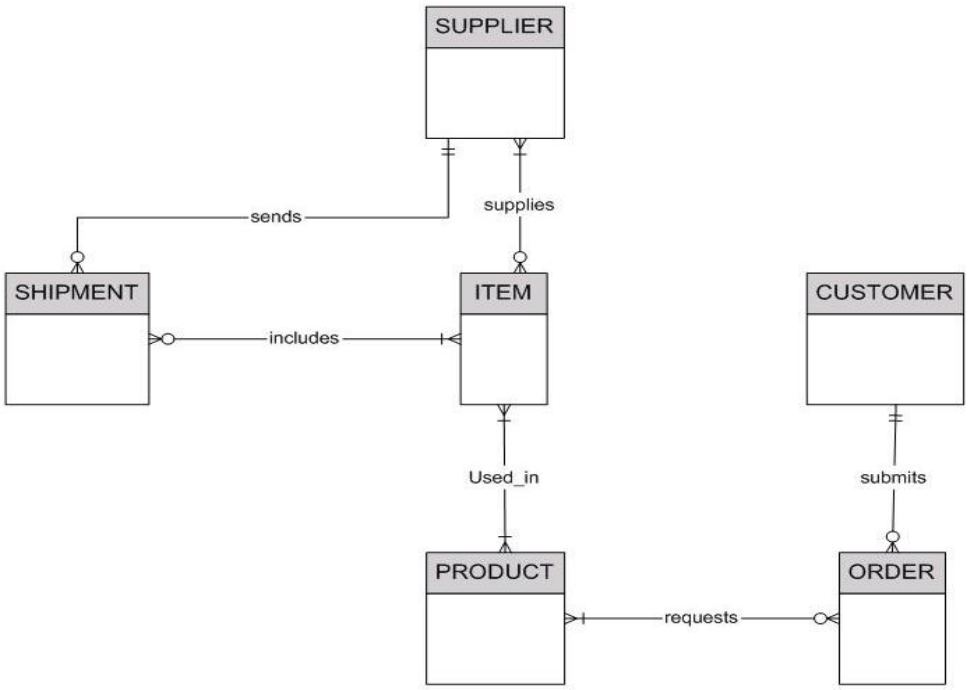


Figure 3.1

Entities

As defined above, an entity is a thing about which the organization wants to hold information. Entities may be physical things, such as:

- Customer;
- Invoice;
- Product;
- Supplier;
- Employee;
- Training course;

Or conceptual things related to the business area, such as:

- Salary grade;
- Sickness history;
- Project.

Entity occurrence

An entity has a number of occurrences. If there are 200 employees in an organization there will be 200 occurrences of the entity 'employee'. Each individual occurrence must have a unique identifier (a key).

For example,

Employee no:	Name:	Address:	Dept:	Salary:	...etc.
A624	Brown, B.J.	Nottingham	Accounts	£25,000	

Is one occurrence of the entity employee with a unique key of employee number

Attributes

All entities have attributes; they are the data, which describes or qualifies the entity. In the entity employee the attributes could be:

- Employee number;
- Employee name;
- Employee address;
- Employee department;
- Employee salary, etc.

In the previous example, 'A646' is the specific value of the attribute employee number for one occurrence of the entity employee.

Relationships

The relationship between two entities describes the way in which an occurrence of one entity is linked to, or influenced by, occurrences of another. For example, a department may have zero, one or many employees, but each employee will relate to only one department (in this example, where the business rule is that an employee only ever works in one department).

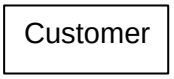
Notation

Entity description. The entity name is always in the singular within a rectangle, which can be 'hard' or 'soft' (with rounded comers) depending on which structured development method is followed.

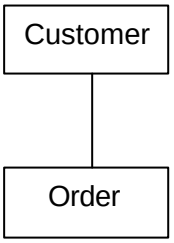
Relationships. A relationship between two entities is shown as a line.

Notation for ER Diagram

Basic Symbols of ERD



An entity is a thing about which the organization wants to hold information. Entities may be physical things



Relationship between two entities

Steps to create a Entity Relationship Diagram

Step 1: Selecting initial entities.

Initial entities can identify by looking at existing files and documents. Where there is a unique key (identifier) this will be a candidate for an entity. Employee has a unique employee number, an invoice a unique invoice number etc.

Step 2: Placing the initial entities in a grid.

An entity grid is shown in Figure 3.2. The analyst must establish where direct relationships exist, that is, no other entity comes between. For example, in customer, order, product scenario the customer receives the product, which might indicate a relationship; however, the customer cannot receive the product without an order, therefore there is no direct relationship between the two (Figure 3.2).

	Book	Brower	Topic	Author	Reserve
Reserve	*X*	*X*			
Author	*X*				
Topic	*X*				
Brower	*X*				
Book					

Figure 3.2

Step 3: Converting the grid into an initial entity relationship diagram

Each entity in the grid can represent by a box on the diagram. A line is drawn between boxes where there is an 'X' in the grid (Figure 3.3).

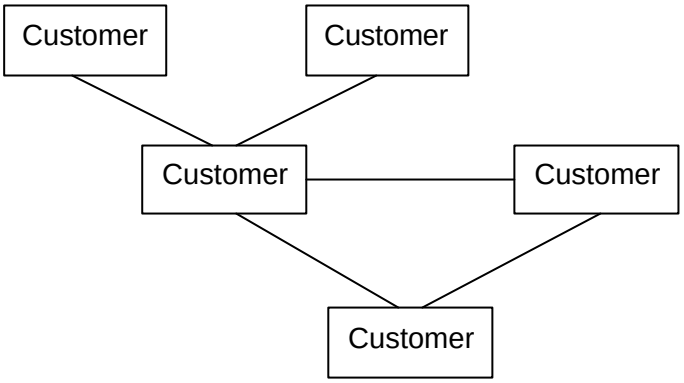


Figure 3.3

Step 4: Determining the degrees of the relationship

Relationships can be of three types:

- One to one
- One to many
- Many to many

Identify and apply relationships to ERD.

Step 5: Identifying the additional characteristics.

There can be many additional relationships. These are

- Name relationships
- Optional relationships
- Exclusive relationships
- Recursive relationships

Identify and Apply these to ERD.

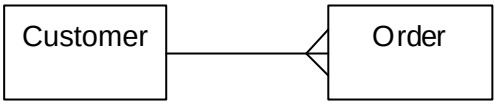
Step 6: Validating the ERD.

Compare with the business case and validate the Database.

Relationships of higher degree

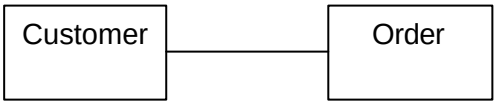
There can be mainly three types of relationships.

One-to-many relationship



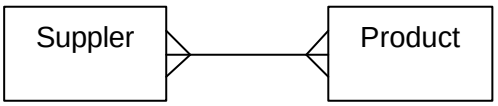
This notation will show the: “One customer can make many orders”

One-to-many relationship



This notation will show the:” One customer can make only one order”

Many-to-many relationship



This notation will show the:” One suppler can supply many products as well as one product can supply by many suppliers ”

Extended ER model

To draw the extended ERD we should identify the additional characteristics.

Name Relationships

The relationships between two entities should be named in both directions. In Figure 3.4, an author is the writer of a book, and the book is written by an author.

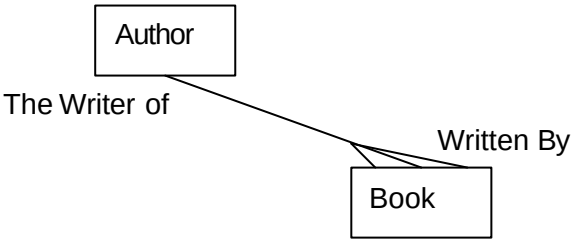


Figure 3.4

Optional Relationships

A book can exist without being issued; the relationship with issue is zero, one or more; however, an issue cannot exist without a book. Some disciplines use a dotted line to show optional relationships and a solid line to show a permanent relationship, as in Figure 3.5. In the figure, there is no optionality between author and book: if details of an author are kept it is because there are books in the library written by that author. A book is always written by an author. Borrower details can be kept without that borrower taking out any books (the optionality is shown by the dotted line); however, issue details must be associated with a borrower (shown by the solid part of the line between issue and borrower). A relationship defines the association between a master entity and a detail entity.

For each master occurrence it should be possible to access all details Optional relationships. for that master. The customer is the master and orders the detail. Given a customer number, all orders for that customer can be accessed. In order/ product the order is the master and product the detail. Given an order number all products on that order can be accessed. An entity can therefore be a master in one relationship and a detail in another (Figure 3.6). The dotted line can indicate optional masters, e.g. an Invoice has a one-to- many relationship with a training course, but training course details can exist without an invoice. An alternative approach, where the dotted line approach is not used, is to place a '0' on the relationship line indicating optional masters (Figure 3.7).

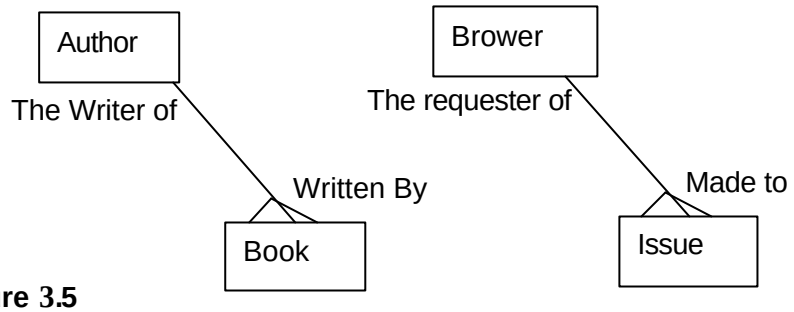


Figure 3.5

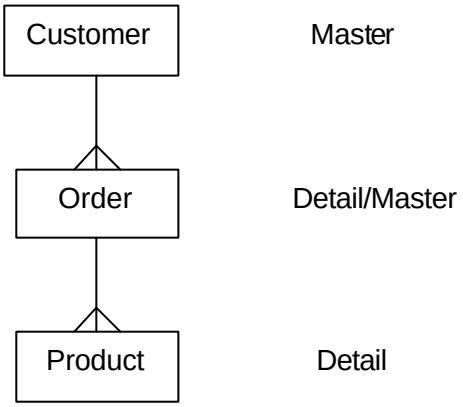


Figure 3.6

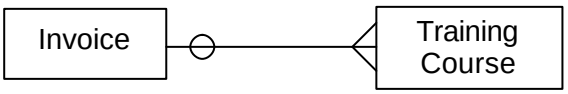


Figure 3.7

aster

Exclusive relationships.

Exclusive relationships describe either/or situation; this is indicated by an arc, which shows exclusive masters or exclusive detail (see Figures 3.8 and 3.9). In Figure 3.8, an employee has either-weekly pay details or monthly pay details. In Figure 3.9, an invoice is sent to either supplier or a customer.

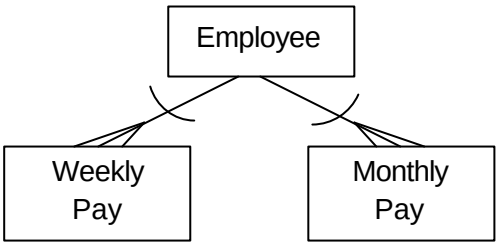


Figure 3.8

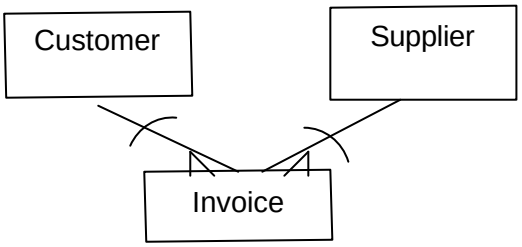


Figure 3.9

Recursive relationships.

There are times when entities have relationships with themselves. In Figure 3.10, for example, each employee reports to a supervisor who is also an employee. Each employee who is a supervisor may supervise many other employees. An optional recursive relationship is shown in Figure 3.10.

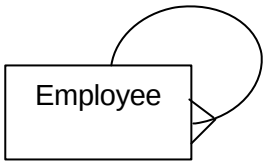


Figure 3.10

CHAPTER 4

Relational Data Model and Languages

Introduction

The relational model defined by E.C. Codd in the early seventies is a theoretical model of a database. The model appealed to the computer science community because of its mathematical basis and to the computing industry at large because of the simple way in which it represented information by the well-understood convention of tables of values.

This session will give an overview/review of the relational model and relational Algebra, Data base constraints and basic overview on Structured Query Language (SQL) as Data Manipulation language.

Concepts in Relational Model

Relational Data Structure

We can view the table to be the basic object of the relational model. Relational model tables conform to the intuitive notion of tables with columns of values and a header name for each column with which we are all familiar. The flowing figure illustrates the components of the relational data structure (or more precisely, the structural part of the relational model):

- Relation
- Attribute and domain
- Tuple and key
- Header and body
- Degree and cardinality

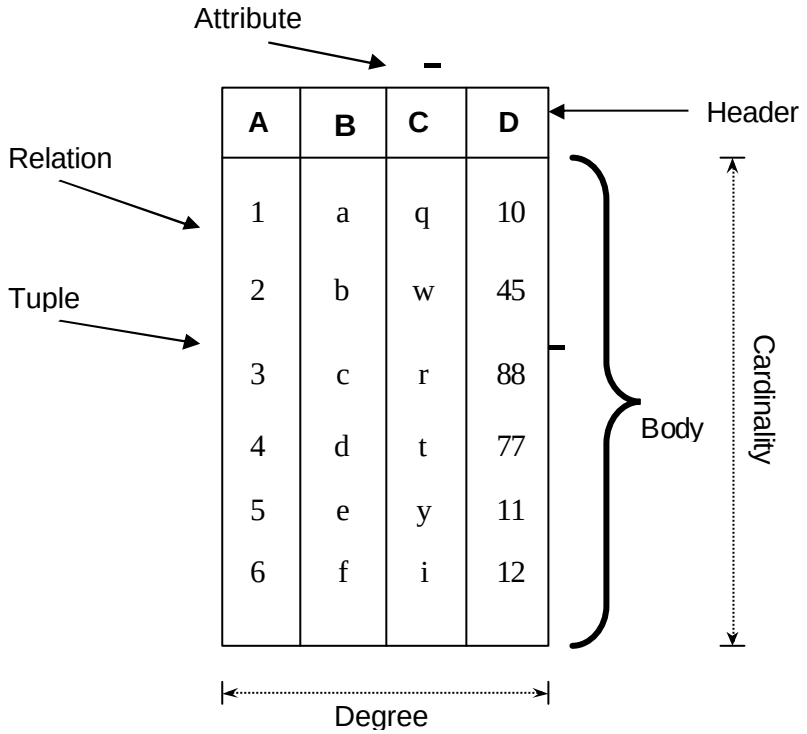
Definitions

- *Database* - Comprised of a collection of table-like structures called relations.
- *Relation* - A relation is a set of tuples defined on a number of attributes.
- *Attribute* - Attributes are like the columns of a conventional table. Each attribute has an attribute name (like the column heading) and attribute values (like column entries).
- *Tuple* - A tuple can be likened to a row in a conventional table. Each tuple is a set of attribute values, one for each attribute of the relation. Furthermore, each tuple has the same number of attribute values.

- *Header and body* - As illustrated, the table (relation) has two distinct parts, a header part and a body part. The header part of the table is a simple mathematical set, including a list of attribute names, one attribute name for each column. Each attribute name must be unique within a table. The body part consists of a number of tuples. The intersection of a tuple with each column of the table holds an attribute value. Each column in the body part of the table holds attribute values corresponding to only one type of attribute. The list of values which attributes in one column can take is referred to as the domain of that attribute.

  We already know the importance of distinguishing between types and occurrences of things. The header part of a relational model table defines a type of entity or relationship. The body of the table contains occurrences of that entity type or relationship, with each tuple corresponding to one occurrence.

  - *Degree - The number of attribute columns of a relation is called the degree of the* relation
  - Cardinality - The number of tuples of a relation is called the cardinality of the relation



Domains and Keys

Domain - This is a set of values of a given type. For example, the domain of an attribute named Supplier Number is the set of all possible supplier numbers, the domain of Shipment Quantities is the set of all integers greater than zero and less than 10,000 (say). Thus domains are pools of values, from which the actual values appearing in attributes are drawn. Every attribute must be defined on exactly one underlying domain, meaning that values of that attribute must be drawn from that domain.

- One aspect of the significance of domains is that domains constrain comparisons. For example, it makes sense to compare two part numbers which make up attributes in two separate relations, but it does not make sense to compare a weight with a quantity.

- Both of these attributes are numbers, but they are different kinds of numbers. The weight and quantity domains would therefore be distinct. We can state that if two attributes draw their values from the same domain, then comparisons - and hence joins, unions and many other operations – involving those two attributes make sense because they are comparing like with like. Thus, one advantage of having the system support domains is that it enables the system to prevent users from making silly mistakes.

Keys -A set of attributes whose values uniquely identify each tuple of a relation.

- Candidate key - Any attribute that satisfies the above definition. A relation may have many keys.
- Primary key and alternate key - Among all the candidate keys of a given relation, one will be chosen to be the primary key, and the others are called alternate keys. Remember that primary key values are used to identify uniquely each tuple within a table. Therefore, the values of a primary key attribute must each be unique within the domain of that attribute.
- Composite key - When more than one attribute column is needed to establish unique identification for tuples within a table, the resulting primary key is referred to as a concatenated primary key.
- Foreign key - These are attributes used to cross-reference tuples using the tuples' primary key values. In other words, a primary key for one table is known as a foreign key in the table into which it is embedded for the purpose of identifying relationship occurrences.

  Foreign keys are used to represent relationships. There are no links or pointers connecting one table to another. In non-relational systems, by contrast, such information is typically represented by some kind of physical link or pointer that is explicitly visible to the user.

Relations

There is only one data structure in the relational data model - the relation. Because the idea of a relation is modeled on a mathematical construct, a relation is a table, which obeys a certain restricted set of rules:

1. Every relation in a database must have a distinct name, a two-dimensional table for Codd is a mathematical set and mathematical sets must be named unambiguously.
2. Every column in a relation must have a distinct name within the relation. Each column of a relation is also a set and hence should also be named unambiguously.
3. All entries in a column must be of the same kind. This is implied from 2.
4. The ordering of columns in a relation is not significant. The head of a relation-its list of column names - is also a mathematical set. Sets in mathematics are not ordered.
5. Each row in a relation must be distinct. In other words, duplicate rows are not allowed in a relation.
6. The ordering of rows is not significant. Since the body of a relation is a set, there should be no implied order in the rows of a relation.
7. Each cell or column/row intersection in a relation should contain only a so-called atomic value. In other words, multiple-values are not allowed in the cells of a relation.

Integrity Constraints

An integrity constraint can be regarded as a condition that all correct states of the database are required to satisfy. Integrity is enforced via integrity rules.

There are three main types of integrity specified in the relational model:

- *Domain integrity* - Domain integrity is concerned with the values that may be contained within a particular column of relations.
- *Entity integrity* - Entity integrity is concerned with the primary keys of relations.
- *Referential integrity* - Referential integrity is concerned with the foreign keys of relations.

Domain Integrity

Domain integrity is concerned with the values that may be contained within a particular column. The domain integrity rule states that every attribute is required to satisfy the constraint that its values are drawn from the relevant domain.

All columns have an implicit domain derived from their data types (for example, a telephone number is made up of 10 numeric digits). However, more explicit domain integrity can also de defined (for example, each telephone number is preceded by the digits 071).

Ideally commands should be available within the database system to allow the user to define domain specifications when the database is created - not all systems currently allow this.

It would also be helpful to have some type of domain constraint. For example, a CHECK clause could be used to specify the gender column as CHECK (gender IN CM', "F)). This would be more useful than just being able to specify the column as type character - which would allow any character.

Entity Integrity

Entity integrity is concerned with primary keys. The entity integrity rule states that every base relation must have a primary key and no component of that primary key is allowed to accept null values.

- *Null values* - Null values here mean that information is missing for some reason (for example, the property is non-applicable), or the value is unknown.
- *The function of primary keys* - Primary keys perform the unique identification function in the relational model.

Primary keys provide the basic tuple-level addressing mechanism because the only system-guaranteed way of pinpointing a specific tuple is by its primary key value. Therefore, each primary key value must identify one and only one tuple.

Implication for primary keys accepting nulls - Suppose a relation Customer included a tuple for which the name value was null. That would be like saying that there was a customer who had no identity.

If that null means property does not apply, then the tuple makes no sense. If it means value is unknown, then confusion will arise and queries using that attribute may get a "don't know" as their reply.

Therefore, the entity integrity rule is sometimes stated that in a relational database, we never record information about something we cannot identify.

Referential integrity

Referential integrity is concerned with foreign keys. The referential integrity rule- states that the database must not contain any unmatched foreign key value. In other words, the referential integrity rule simply says that if references A. then A must exist.

- *Referencing vs. referenced* - A foreign key is an attribute of one relation R2 whose values are required to match those of the primary key of some relation Rl. We refer to the relation that contains the foreign key as the referencing relation and the relation that contains the corresponding primary key as the referenced or target relation. A given relation can, of course, be both a referenced relation and a referencing relation.
- *Domain requirement* - A foreign key value represents a reference to the tuple containing the matching primary key value (the referred tuple or target tuple). Therefore, a given foreign key and the corresponding primary key should be defined on the same underlying domain.
- *Possible cases for null values* - Unlike primary keys, foreign keys do sometimes have to accept null values. In most cases, the reasons for using nulls are likely to fall in the value does not exist category, rather than the value unknown category. For example, in the case of the department-and- employees database, it might be possible for some employee to be currently assigned to no department at all.

Maintaining Referential Integrity in relational Database

Referential integrity should maintain in two situations as follows.

- Deletion - What should happen on an attempt to delete the target of a foreign key reference? For example, an attempt to delete a customer for which there exists at least one matching Order?

  In general there are at least three possibilities:

  1. *Restricted* - the delete operation is restricted to the case where there are no such matching orders (it is rejected otherwise).

     For example, the deletion of a parent row will be disallowed if there are any dependent rows.
  2. *Cascades* - the delete operation cascades to delete those matching orders also.

     For example, if a parent row is deleted, then the system will automatically delete the dependent rows.
  3. Nullifies - the foreign key is set to null in all such matching orders and the customer is then deleted.

     For example, when a parent row is deleted, all dependent foreign keys are set to null.
- Updating - What should happen on an attempt to update the primary key of the target of a foreign key reference? For example, an attempt to update the name for a customer for which there exists at least one matching order.

  Again there are 3 possibilities:

  1. Restricted - the update operation is restricted to the case where there are no such matching orders (it is rejected otherwise).

     For example, if an update command changes the primary key of a parent row and if dependants exist for that row, the update is disallowed.
  2. Cascades - the delete operation cascades to update those matching orders also.

     For example, if an update commands changes the primary key of a parent row, all of its dependent rows will have their foreign key updated to the same value.
  3. Nullifies - the foreign key is set to null in all such matching orders and the customer is then updated.

     For example, when an update command changes the primary key of a parent row, all of its dependent rows will have their foreign key set to null.

Relational Algebra

The manipulative part of the relational model defines the set of things, which can be done to relational databases. Relational algebra and relational calculus are the two main alternative ways of expressing the manipulative part of the relational model.

Relational algebra is addressed in this session. Relational algebra is basically a set of operations, which can be applied to a relational database in order to manipulate and access the data contained in tables.

Relational algebra (RA) consists of a collection of high-level operators that operate on relations. In his original paper on the relational model, Codd introduced eight such basic operators which could be used to manipulate data within the body (relation) parts of tables of a relational database.

These eight operators fall into two groups of four:

1. *The traditional set operations* - union, intersection, difference and Cartesian product
2. *The special relational operations* - restrict, project, join and divide.

These basic operators are all incorporated into the standard, international relational database-language, Sequential Query Language (SQL). Note, however, that SQL supports other Data Manipulation Language (DML) operators beyond those just described and that it also supports Data Definition Language (DDL) operators.

Property of closure - It is important that the results of using the above operators on tables must themselves be tables. This is because these operators can be used sequentially in various combinations to obtain desired results. Thus each operation on completion must leave data as a table (or tables) for the next operator to use. This property, which all the above operators must have, is referred to as closure.

Definition of Relational Algebra (from Date)

Restrict - Builds a relation consisting of all tuples from a specified relation that satisfy a specified condition.

Project - Builds a relation consisting of all specified attributes from a specified relation.

Product - Builds a relation from two specified relations consisting of all possible combinations of tuples, one from each of the two relations.

Union - Builds a relation consisting of all tuples appearing in either or both of two specified relations.

Intersect - Builds a relation consisting of all tuples appearing in both the first and the second of two specified relations.

Difference - Builds a relation consisting of all tuples appearing in the first and not the second of two specified relations.

Join - Builds a relation from two specified relations consisting of all possible combinations of tuples, one from each of the two relations, such that the two tuples contributing to any given combination satisfy some specified condition.

Divide - Takes two relations, one binary and one unary, and builds a relation consisting of all values of one attribute of the binary relation that match (in the other attribute) all values in the unary relation.

Structured Query Language

In this section we shall reflect the relational data model against the contemporary practice of relational database systems. Contemporary practice is primarily centered around a language known as SQL (Structured Query Language). SQL was originally designed as a query language based on the relational calculus. The current specification of SQL however is a lot more than simply a query language. It is more accurately described as being a database sub- language. Indeed, this database sub-language is becoming the standard interface to relational and non-relational DBMS.

SQL has its origins in work done at the IBM research laboratory in San Jose, California during the early 1970s. Here a prototype implementation of relational concepts named System/R was built. This early RDBMS embodied a language then known as SEQUEL. This is the reason why many people still refer to the SQL language by this term rather than the acronym.

During the years 1973 to 1979 IBM researchers published a great deal of material about the development of System/R in academic journals. This period was characterized by intense discussion about the validity of RDBMS at conferences and seminars both in the US and in Europe. IBM was however undoubtedly slow to see the commercial relevance of relational systems. It fell to the ORACLE Corporation, founded in 1977; to first exploit successfully in the commercial world the ideas underlying the relational data model.

ORACLE was, and is, an SQL-based RDBMS. Many other vendors also produced systems that support SQL. For these reasons, in 1982 the American National Standards Committee gave its database committee (X3H2) the remit to develop a standard Relational Database Language (RDL). This committee finally produced a definition for a standard SQL syntax in 1986, based primarily on the IBM and Oracle dialects of SQL (ANSI, 1986). The International Standards Organization followed suit with a publication of much the same standard in 1987 (ISO, 1987). This standard is also known as SQL I. The original ANSI document specifies two levels for SQL I: level one and level two. Level two is the complete SQL language. Level one is a subset of level two originally intended to act as the intersection of existing implementations.

Following its publication, a number of criticisms were made of the ANSI/ISO standard, most notably by database personalities such as E.F. Codd (1988a, 1988b) and C. Date (1987). Many people viewed the standard as suffering from being the lowest common denominator amongst implementations. Others saw the language have more serious defects, particularly in its ability to address relational construe In response to some of these criticisms, an addendum to the standard was published in 1989 by ANSI, primarily addressing a number of integrity enhancement features (ANSI, 1989a). Much of the material in this addendum was include in a working draft of a proposed second version to the standard also published.

SQL is normally divided into three major parts: data definition, data manipulation and data control. Included within its idea of data definition is the issue of data integrity. And SQL contains a number of important facilities to control data access.

Data Definition

Data Definition is define the Data base structure. The initial activity is the creating the table (Relation)

The CREATE TABLE statement.

Syntax:

CREATE TABLE <table Name>

(<Column Name><Data type>(<Length>),

(<Column Name><Data type>(<Length>),
……..)

Example :

CREATE TABLE Modules

(ModuleName CHARACTER(15),

courseCode CHARACTER(3),
StaffNo INTERGER)

In above SQL statement user should specify the four things.

1. Name for the table
2. Name of the columns in the table
3. Data Type of the each column
4. Maximum length of the column

NOT NULL AND UNIQUE

Any column in a table can be specified as being NOT NULL. This means that the user is then unable to enter null values into that column. The default specification for a column is null. That is, null values are allowed in a column. Any column can also be defined as being UNIQUE. This clause prohibits the user from entering duplicate values into column. The combination of NOTNULL and UNIQUE can be used to define the characteristics of a primary key.

For example:

CREATE TABLE Modules

(moduleName CHARACTER(15) NOT NULL UNIQUE,
level SMALLINT,
courseCode CHARACTER(3),
staffNo INTEGER)

DEFAULT VALUES

A clause can be added to a column definition specifying the value that a column should take in response to incomplete information being entered by the user. For instance, a DEFAULT <value> specification can be added to the level column for modules indicating that the default level should be I.

For example:

```
CREATE TABLE Modules
    (moduleName CHARACTER(15) NOT NULL UNIQUE,
    level SMALLINT DEFAULT I,
    courseCode CHARACTER(3),
    staffNo INTEGER)
```

DROP TABLE

Table definitions can be created and table definitions can be deleted. To remove a table from the database we use the following command:

Syntax:

```
DROP TABLE <table name>
```

For example:

```
DROP TABLE Modules
```

MODIFYING TABLES

To modify the structure of a database without impacting on the users or application programs which access this database. In practice, SQL-based products support only a limited form of data independence. The database administrator is allowed to add an extra column to a table, modify the maximum length of an existing column, or drop a column from a table. Each operation is specified using the ALTER TABLE command.

For instance:

```
ALTER TABLE Lecturers
ADD COLUMN roomNo SMALLINT

ALTER TABLE Lecturers
ALTER COLUMN staffName VARCHAR(20)

ALTER TABLE Lecturers
DROP COLUMN staffName
```

Queries and updates in SQL

SQL was originally designed primarily as a means for extracting data from a database. Such extraction is accomplished through use of the select command: a combination of the restrict, project, and join operators of the relational algebra.

Simple Retrieval

Simple retrieval is accomplished by a combination of the select, from and where clauses:

Syntax:

SELECT <attribute name>, <attribute2 name>, ...

FROM -stable name>

[WHERE <condition>]

The select clause indicates the table columns to be retrieved. The “FROM” clause defines the tables to be referenced. The “WHERE” clause indicates a condition or conditions to be satisfied. The following command, for instance, is a direct analogue of the relational algebra select or restrict. The asterisk ' * ' acts as a wildcard. That is, all the attributes in the table are listed:

```
SELECT *
FROM Modules
```

Note, that the example above has no “WHERE” clause. The “WHERE” clause is optional. we omit the where clause then all the rows of a table are considered by the query. The addition of a where clause restricts the retrieval to a set of rows matching a given condition:

```
SELECT *
FROM Modules
WHERE moduleName=’Reletinal Data base’
```

CHAPTER 5

Relational Database Design

Introduction

In this chapter we consider Codd's original ideas on normalization while also

Describing a graphic technique used for designing fully normalized schema. We particularly emphasize the use of normalization as a bottom-up technique for relational data base design.

Normal forms

In his seminal paper on the relational data model, E. F. Codd formulated a number of design principles for a relational database (Codd 1970). These principles were originally formalized in terms of three normal forms: first normal form, second normal form and third normal form. The process of transforming a database design through these three normal forms is known as normalization. By the mid-1970s third normal form was shown to have certain inadequacies and a stronger normal form, known as Boyce-Codd normal form was introduced (Codd 1974). Subsequently Fagin introduced fourth normal form and indeed fifth normal form (Fagin 1977, 1979).

Why Normalise?

Suppose we are given the brief of designing a database to maintain information about students, modules and lecturers in a university. An analysis of the documentation used by the administrative staff gives us the following sample data set

with which to work. If we pool all the data together in one table as below, a number of problems, sometimes called file maintenance anomalies, would arise in maintaining this data set.

Modules						
Module Name	***Staff No***	***Staff Name***	***Student No***	***Student***	***Ass Grade***	***Ass Type***
Relational Database Systems	234	Lee T	3468	Smith S	B3	Cwk1
Relational Database Systems	234	Lee T	3468	Smith S	B1	Cwk2
Relational Database Systems	234	Lee T	3778	Jones S	B2	Cwk1
Relational Database Systems	234	Lee T	3488	Patel P	B1	Cwk1
Relational Database Systems	234	Lee T	3488	Patel P	B3	Cwk2
Relational Database Design	234	Lee T	3468	Smith S	B2	Cwk1
Relational Database Design	234	Lee T	3468	Smith S	B3	Cwk2
Deductive Databases	345	Evans	3478	Smith J	A1	Exam

Figure 5.1

1. What if we wish to delete student 3468? The result is that we lose some valuable information. We lose information about deductive databases and its associated lecturer. This is called a deletion side effect.
2. What if we change the lecturer of deductive databases to V Konstantinou? We need to update not only the staff-Name. But also the staff-No for this module. This is called an update side effect.
3. What if we admit a new student on to a module, say student-No 3898? We cannot enter a student record until a student has had at least one assessment. This is known as an insertion side effect.
4. The size of our sample file is small. One can imagine the seriousness of the file maintenance anomalies mentioned above multiplying as the size of the file grows. The above structure is therefore clearly not a good one for the data of this enterprise. Normalization is a formal process whose aim is to eliminate such file maintenance anomalies.

Stages of Normalization

Normalization is carried out in the following steps;

1. Collect the data set - the set of data items.
2. Transform the unnormalised data set into tables in first normal form.
3. Transform first normal form tables to second normal form.
4. Transform second normal form tables to third normal form.

Occasionally, the data may still be subject to anomalies in third normal form. In this case, we may have to perform further steps;

1. Transform third normal form to Boyce-Codd normal form.
2. Transform third normal form to fourth normal form.
3. Transform fourth normal form to fifth normal form.

The process of transforming an unnormalised data set into a fully normalized (Third normal form) database is frequently referred to as a process of non-loss decomposition. This is because we continually fragment our data structure into more and more tables without losing the fundamental relationships between data items.

Functional dependencies

Determinacy/Dependency

Normalization is the process of identifying the logical associations between data items and designing a database, which will represent such associations, but without suffering the file maintenance anomalies discussed in section 5.1. The logical associations between data items that point the database designer in the direction of a good database design are referred to as determinant or dependent relation ships. Two data items, A and B, are said to be in a determinant or dependent relationship if certain values of data item B always appear with certain values of data item A. Determinacy/dependency also implies some direction in the association. If data item A is the determinant data item and B the dependent data item then the direction of the association is from A to B and not vice versa.

There are two major types of determinacy or its opposite dependency: functional (single-valued) determinacy and non-functional (multi-valued) determinacy. We introduce here the concept of functional determinacy.

Data item B is said to be functionally dependent on data item A if for every value of A there is one, unambiguous value for B. In such a relationship data item A is referred to as the determinant data item, while data item B is referred to as the dependent data item. Functional determinacy is so-called because it is modeled on the idea of a mathematical function. A function is a directed one-to-one mapping between the elements of one set and the elements of another set. In a university personnel database, staff No and Staff Name are in a functional determinant relationship. Staff No is the determinant and Staff Name is the dependent data item. This is because for every Staff No there is only one associated value of Staff Name. For example, 345 may be associated with the value J.Smith. This does not mean to say that we cannot have more than one member of staff named J.Smith in our organization. It simply means that each J.Smith will have a different Staff No. Hence, although there is a functional determinacy from Staff No to Staff Name the same is not true in the opposite direction – Staff Name does not functionally determine Staff No. Staying with the personnel information, Staff No will probably functionally determine Department Name. For every member of staff there is only one associated

department or school name which applies. A member of staff cannot belong to more than one department or school at any one time.

Determinacy Diagrams

A diagram, which documents the determinacy or dependency between data items we shall refer to as a determinacy or dependency diagram. Data items are drawn on a determinacy diagram as labeled ovals, circles or bubbles. Functional dependency is represented between two data items by drawing a single-beaded arrow from the determinant data item to the dependent data item. For example, figure 5.4(A) represents a number of functional dependencies as diagrams.

First Normal Forms

Unnormalised Data Set to First Normal Form

The data set represented in tabular form in section 5.2 is said to be an unnormalised data set. This can be seen, for instance, if we choose the data item module Name as the key of this data set and underline it to indicate this. Realistically, if we remove redundant information, we should represent the information as follows;

Unnormalised data set:

Modules						
Module Name	***Staff No***	***Staff Name***	***Student No***	***Student***	***Ass Grade***	***Ass Type***
Relational Database Systems	234	Lee T	3468	Smith S	B3	Cwk1
					B1	Cwk2
			3778	Jones S	B2	Cwk1
			3488	Patel P	B1	Cwk1
					B3	Cwk2
Relational Database Design	234	Lee T	3468	Smith S	B2	Cwk1
					B3	Cwk2
Deductive Databases	345	Evans	3478	Smith J	A1	Exam

Figure 5.2

A given cell of the table for the attributes Student No, Student Name, Ass Grade and Ass Type contain multiple values. Examining the table above we see that Student No, Student Name, Ass Grade and Ass Type all repeat with respect to Module Name.

(A relation is in first normal form if and only if every non-key attribute is functionally dependent upon the primary key.)

The attributes Student No, Student Name, Ass Grade and Ass Type are clearly not functionally dependent on our chosen primary key Module Name. The attributes Staff No and Staff Name clearly are. This means that we form two tables: one for the functionally dependent attributes, and one for the non-dependent attributes. We declare a compound of Module Name, Student No and Ass Type to be the primary key of this second table.

1st normal form tables:

Modules		
Module Name	***Staff No***	***Staff Name***
Relational Database Systems	234	Lee T
Relational Database Design	234	Lee T
Relational Database Design	234	Lee T
Deductive Databases	345	Evans

Assessments				
Module Name	***Student No***	***Ass Type***	***Student Name***	***Ass Grade***
Relational Database Systems	3468	Cwk1	Smith S	B3
Relational Database Systems	3468	Cwk2	Smith S	B1
Relational Database Systems	3778	Cwk1	Jones S	B2
Relational Database Systems	3488	Cwk1	Patel P	B1
Relational Database Systems	3488	Cwk2	Patel P	B3
Relational Database Design	3468	Cwk1	Smith S	B2
Relational Database Design	3468	Cwk2	Smith S	B3
Deductive Databases	3478	Exam	Smith J	A1

Second Normal Forms

First Normal Form To Second Normal Form

To move from first normal form to second normal form we remove part-key dependencies. This involves examining those tables that have a compound key and for each non-key data item in the table asking the question: can the data item be uniquely identified by part of the compound key?

A relation is in second normal form if and only if it is in IMF and every non-key attribute is fully functionally dependent on the primary key.

Take, for instance, the table named Assessments. Here we have a 3-part compound key Module Name, student No and ass Type. We ask the question above for each of these data items in relation to the non-key data items Student Name and Ass Grade. Clearly we need all the items of the key to tell us what the assessment grade is. Module Name, however, has no influence on the Student Name. Student No alone determines Student Name. Hence, we break out the determinant and dependent data items in to their own table. This leads to a decomposition of the tables as follows:

2nd normal form tables:

Modules

Module Name	***Staff No***	***Staff Name***
Relational Database Systems	234	Lee T
Relational Database Design	234	Lee T
Relational Database Design	234	Lee T
Deductive Databases	345	Evans

Assessments

Module Name	***Student No***	***Ass Type***	***Ass Grade***
Relational Database Systems	3468	Cwk1	B3
Relational Database Systems	3468	Cwk2	B1
Relational Database Systems	3778	Cwk1	B2
Relational Database Systems	3488	Cwk1	B1
Relational Database Systems	3488	Cwk2	B3
Relational Database Design	3468	Cwk1	B2
Relational Database Design	3468	Cwk2	B3
Deductive Databases	3478	Exam	A1

Students

Student No	***Student Name***
3468	Smith S
3778	Jones S
3488	Patel P
3478	Smith J

Third Normal Forms

Second Normal Form to Third Normal Form

To move from second normal form to third normal form we remove interdata dependencies. To do this we examine every table and ask of each pair of non-key data items: is the value of field “A” dependent on the value of field “B”, or vice versa? If the answer is “yes” we split off the relevant data items into a separate table.

A relation is in third normal form if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

The only place where this is relevant in our present example is in the table called Modules. Here, staff No determines Staff Name. Staff Name is hence transitively dependent on Module Name. Staff No is therefore asking to be a primary key. Hence, we create a separate table to be called Lecturers with Staff No as the primary key. This is illustrated below:

3rd normal forms tables:

Modules	
Module Name	***Staff No***
Relational Database Systems	234
Relational Database Design	234
Relational Database Design	234
Deductive Databases	345

Lecturers	
Staff No	***Staff Name***
234	Lee T
345	Evans

Assessments			
Module Name	***Student No***	***Ass Type***	***Ass Grade***
Relational Database Systems	3468	Cwk1	B3
Relational Database Systems	3468	Cwk2	B1
Relational Database Systems	3778	Cwk1	B2
Relational Database Systems	3488	Cwk1	B1
Relational Database Systems	3488	Cwk2	B3
Relational Database Design	3468	Cwk1	B2
Relational Database Design	3468	Cwk2	B3
Deductive Databases	3478	Exam	A1

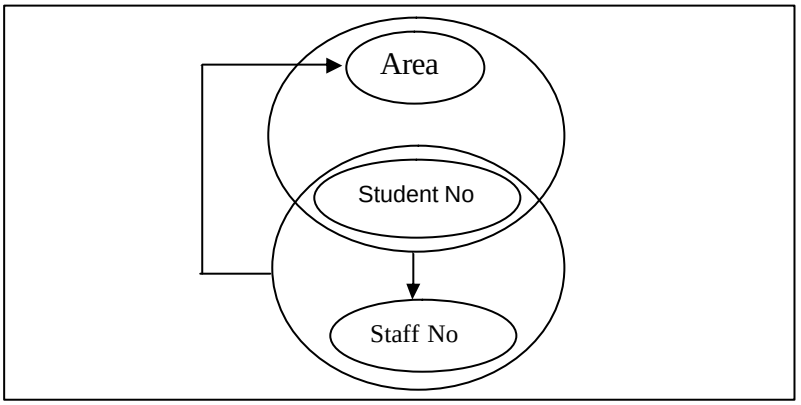
Students	
Student No	***Student Name***
3468	Smith S
3778	Jones S
3488	Patel P
3478	Smith J

Boyce-Codd Normal Form (BCNF)

This is a stronger normal form than 3NF and is designed to cover anomalies that arise when there is more than one candidate key in some set of data requirements. Suppose we have introduced a scheme of majors and minors into our degree schemes at a university. The business rules relevant to that part of these domain-covering majors are listed below:

1. Each student may major in several areas.
2. A student has one tutor for each area.
3. Each area has several tutors but a tutor advises in only one area.
4. Each tutor advises several students in an area.

A diagram incorporating all these business rules is illustrated in figure 5.3 On the basis of these business rules a schema is produced in 3NF represented in the bracketing notation below:



Figurer 5.3

The Bracketing Notation

To represent the relational schema in an implementation-independent form we use a notation sometimes known as the bracketing notation. This is shorthand for a full schema definition as described.

We list a suitable mnemonic name for the table first. This is followed by a list of data items or column names delimited by commas. It is conventional to list the primary key for the table first and underline this data item. If the primary key is made up of two or more attributes, we underline all the component data items. For instance, the third normal form tables for our university example would look as follows:

Modules (Module Name, Staff No)
Lecturers (Staff No, Staff Name)
Assessments (Module Name, Student No, Ass Type, Ass Grade)
Students (Student No, Student Name)

A set of sample data is provided in the table below.

Majors

Student No	*Area*	*Staff No*
123456	Computer Science	234
234567	Information Systems	345
123456	Software Engineering	456
234567	Graphic Design	678
345678	Information System	567

This schema is in 3NF because there are no partial dependencies and no interdata dependencies. However, anomalies will still arise when we come to update this relation. For instance:

1. Suppose student 123456 changes one of her majors from computer science to information systems. Doing this means that we lose information about staff No 234 tutoring on computer science. This is an update anomaly.
2. Suppose we wish to insert a new row to establish the fact that staff No 789 tutors on computer science. We cannot do this until at lest one student takes this area as their major. This is an insertion anomaly.
3. Suppose student 345678 withdraws from the university. In removing the relevant row we lose information about staff No 567 being a tutor in the area of information systems. This is a deletion anomaly,

These anomalies occur because there are two overlapping candidate keys in this problem. R. F. Boyce and E. F. Codd identified this problem and proposed a solution in terms of a stronger normal form known as BCNF. A relation is in BCNF if every determinant is a candidate key. The schema above can be converted into BCNF in one of two ways. The two schemas are presented in bracketing notation below:

Schema 1:

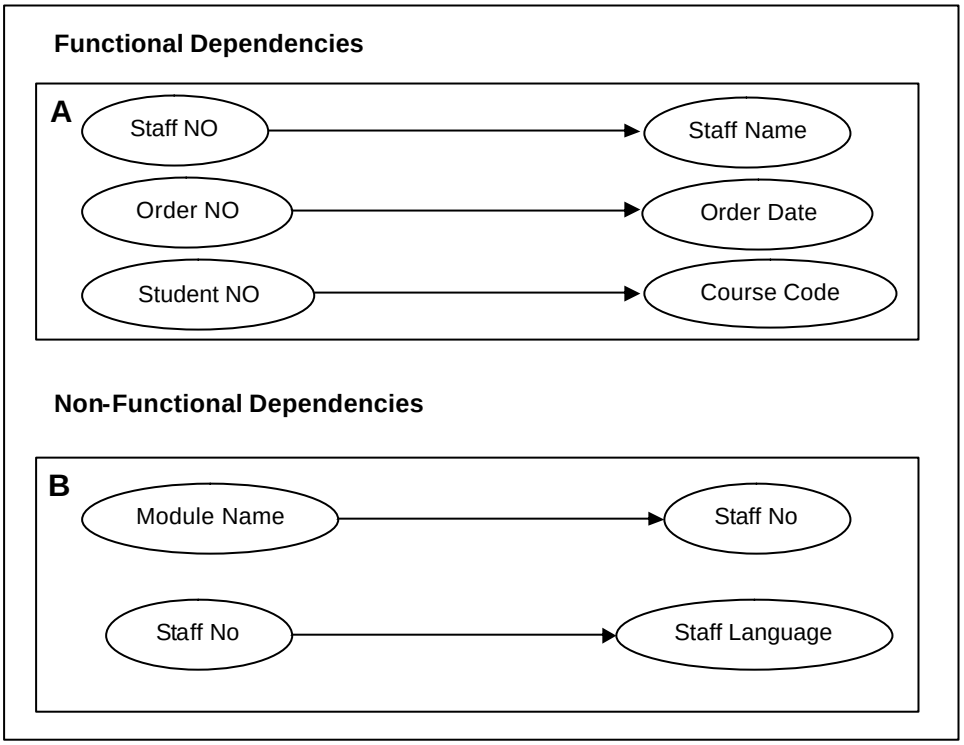
Student Tutors (Student No, Staff No)
Tutor Areas (Staff No, Area)

Schema 2:

Student Tutors (Student No, Area)
Tutor Areas (Staff No, Area)

Multi Valued Dependencies (Non-Functional Dependencies)

Not all dependencies can be modeled in terms of functions. It is for this reason that we need to introduce the idea of a non-functional dependency. Data item "B" is said to be non-functionally dependent on data item "A" if for every value of data item "A" there is a delimited set of values for data item "B". The mapping is no longer functional because it is one to many.



Figurer 5.4 (Determinacy Diagrams)

Let us assume that the university maintains a list of languages relevant to the organization. The university wishes to record which members of staff have which language skills. Clearly the relationship between staff Nos and languages is not a functional determinacy. Many staff members may just have one language, but some will have two or more languages. Also, each language, particularly in the case of European languages such as English, French and German is likely to be spoken by more than one staff member.

Therefore, staff No and staff Language is in a non-functional or multi-valued determinacy. In other words, for every staff No we can identify a delimited set of language codes, which apply to that staff member.

Multi valued or non-functional dependency is indicated by drawing a double-headed arrow from the determinant to the dependent data item. Figure 5.4(B) represents two non-functional relationships as determinacy diagrams.

Forth Normal Forms

To move from BCNP to fourth normal form we look for tables that contain two or more independent multi-valued dependencies. Multi-valued dependencies are fortunately scarcer than part-key or inter-data dependencies. We examine here an example adapted from Kent (1983).

Suppose we wish to design a personnel database for the Commission of the European Community, which stores information about an employee's skills and the languages an employee speaks. An employee is likely to have several skills (e.g. typing, word-processor operation, spreadsheet operation), and most employees are required to speak at least two European languages. Our first attempt at a design for this system might aggregate all the data together in one table as below:

EUEmployees

Employee No	*Skill*	*Language*
0122443	Typing	English
0122443	Typing	French
0122443	Dictation	English
0221133	Typing	German
0221133	Dictation	French
0332222	Typing	French
0332222	Typing	English

However, we wish to add the restriction that each employee exercises skill and language use independently. In other words, typing as a skill is not inherently linked with the ability to speak a particular language. Under fourth normal form these two relationships should not be represented in a single table as in figure 5.5(A). This is evident when we draw the determinacy diagram as in figure 5.5(B). Having two independent multi-valued dependencies means that we must split the table into two as below:

EUEmployees

Employee No	*Skill*
0122443	Typing
0122443	Dictation
0221133	Typing
0221133	Dictation
0332222	Typing

EUEmployees

Employee No	*Language*
0122443	English
0122443	French
0221133	German
0221133	French
0332222	French

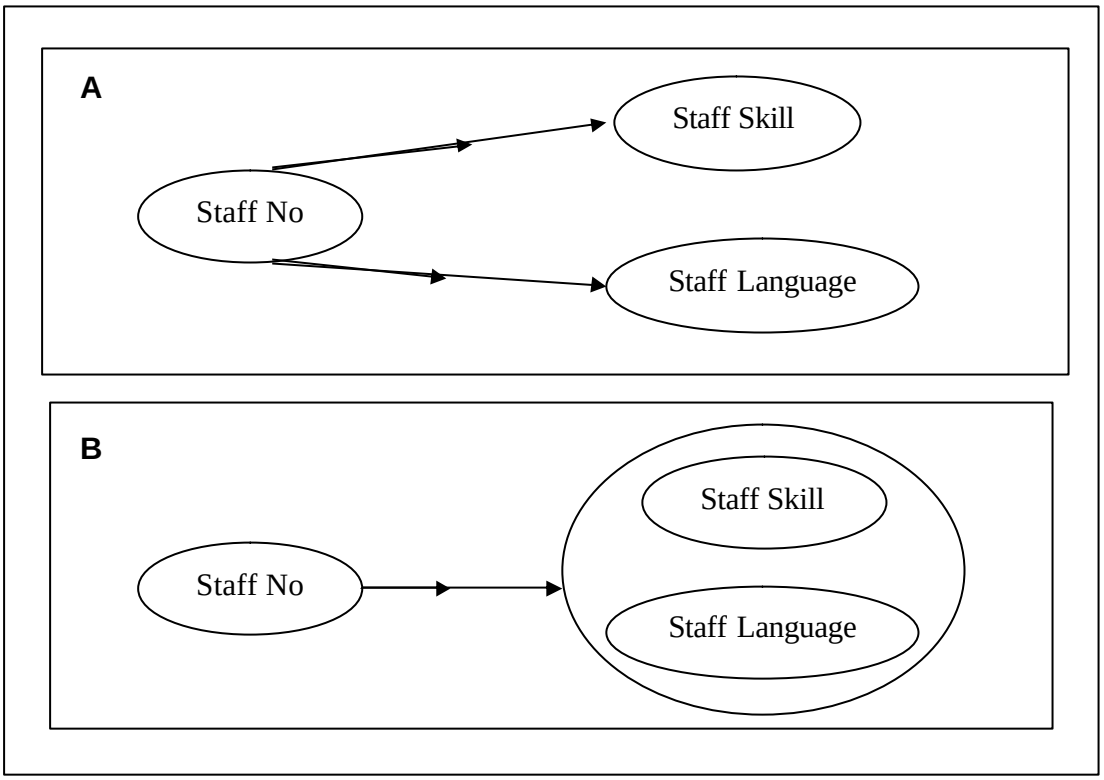

A
Staff Skill
Staff No
Staff Language
B
Staff Skill
Staff No
Staff Language

Figure 5.5

Join Dependencies:

Fifth Normal Form

Generally speaking, a fourth normal form table is in fifth normal form if it cannot be non-loss decomposed into a series of smaller tables. Consider the table below which stores information about automobile agents, automobile companies and automobiles:

Outlets

Agent	*Company*	*Automobile*
Jones	Ford	Car
Jones	Vauxhall	Van
Smith	Ford	Van
Smith	Vauxhall	Car

If agents represent companies, companies make products, and agents sell products, then we might want to record which agent sells which product for which company. To do this we need the structure above. We cannot decompose the structure because although agent Jones sells cars made by Ford and vans made by Vauxhall he does not sell Ford vans or Vauxhall cars.

Fifth normal form concerns interdependent multi-value dependencies, otherwise known as join dependencies.

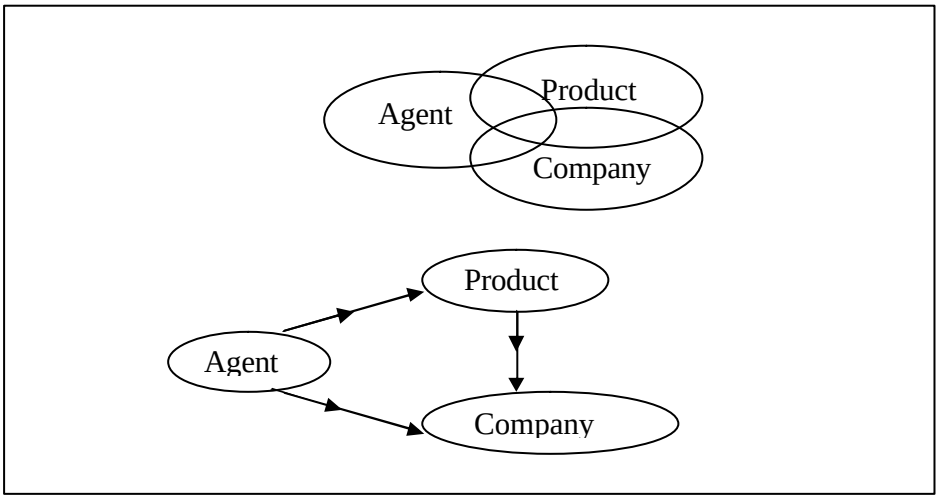


Figure 5.6 (fifth normal form)

Inclusion dependency

Inclusion dependency can be regarded as generalization of referential integrity constrains. This will consent above the values papering in one attribute. But with in the to tables.

Bonus table:

Emp_No	Bonus
0001	1000
0002	2500
0003	1500

Employee master table:

Emp_No	Emp_Name	Address
0001	Jones S	London
0002	Smith J	France
0003	Kert R	Germany

According to inclusion dependency rules the attribute values of bonus file (Emp_No) must be sub set of the values appearing in employee master file. Emp_No is in the bonus file need not to become a foreign key and also emp_No attribute in employee master file need not to be candy date key according to the integrity constraints.

CHAPTER 6

Query processing and optimization

Introduction

In this chapter we will discuses the Query management of DBMS kernel. Under Query management the main topics of Query processing and optimization will discuss and some other functions also.

What is Database Query?

A database Query is the function, which use to retrieve or Update data stored in the database. To perform useful activity with a database we need two types of function: Update and Query functions. Update functions cause changes to data and Query functions will extract data from the database.

We can constrict a query function with the use of Data manipulation Language (DML) Such as SQL. The database query will process by the Query Language Processor in the Kernel of the database system.

Transaction and system concepts

In a multi-user database system the events that cause changes to a database or retrieve data from the database are called transactions. A transaction may define as a logical unit of work. It normally corresponds to some coherent activities performed by an individual, group, organization or software process on a database. In practice this can constitute an entire program or a set of commands, which accesses or updates some database.

To remain an accuracy of the database, any transaction such as the one above should demonstrate the properties of atomicity, consistency, isolation and durability (sometimes abbreviated to ACID):

1. Atomicity. Since a transaction consists of a collection of actions, the DBMS should ensure that either all the transaction is performed or none of it is performed.
2. Consistency. All transactions must preserve the consistency and integrity of the database. Operations performed by an updating transaction, for instance, should not leave the database in an inconsistent or invalid state.
3. Isolation. While a transaction is updating shared data, that data may be temporarily inconsistent. Such data must not be made available to other transactions until the transaction in question has finished using it. For example in airline reservation system, the data structures that contain information about connecting flights should not be available for update by other transactions until this transaction has finished using the information.
4. Durability. When a transaction completes, the changes made by the transaction should be durable. That is, they should be preserved in the case of hardware or software failure.

DBMS ensure the database integrity by maintaining the ACID properties of the database transaction

A single transaction can contend one or more Query functions.

Stages of Query processing

We can identify main four stagers of query processing. Those are:

1. Convert the query into a more suitable internal form. First, the original query is converted into some internal representation, which is more suitable for machine manipulation.
    i. The typical forms used for this representation are:
        1. Query tree;
        2. Relational algebra.
2. Convert to a more efficient canonical form. This internal representation is further converted into some. Equivalent canonical form which is more efficient, making use of well-defined transformation rules
3. Choose set of candidate low-level procedures, using statistics about the database
    i. Low-leva) operation
    ii. Implementation procedure
    iii. Cost formula
4. Generate query plans and choose the best (cheapest) plan by evaluating the cost formulae

Query Optimization

Query optimization is an important component of a modern relational database system. When processing user queries, the query optimizer transforms them into a set of low level operations, and decides in what order these operations are to be executed in the most efficient way.

In a non-relational database system, any optimization has to be carried out manually. Relational database systems, however, offer a system-managed optimization facility by making use of the wealth of statistical information available to the system. The overall objective of a query optimizer is to improve the efficiency and performance of a relational database system.

Effects of Optimization - An Example

In order to appreciate the effects of query optimization on database performance, we use a simple example as an illustration.

Consider the following 3 tables: Student, Lending & Book. The attributes highlighted are the keys for the relevant relations:

Student = (**Stud_No**, Stud_Name, Gender, Address)

Lending = (**Lending_No**, Stud_No, Book_No)

Book = (**Book_No**, Title, Author, Edition)

Consider the following query:

Retrieve the names of students who have borrowed the book B1.

This query can be expressed in SQL:

Select Distinct Student.Stud_Name

From Student, Lending

Where Student.Stud_No = Lending.Stud_No

And Lending.Book_No='B1'

We further make the following two assumptions:

The database contains 100 students and 10,000 lendings, of which only 50 an for book B1.

It is possible to hold up to 50 tuples in memory, without having to write back to disk.

Query Execution Plan A - No Optimization

In this first approach, the operations required by the query are performed in the sequence:

Join-Select-Project

We calculate the number of database accesses (tuple I/O operations) occurred in each operation.

1. *Join* relations Student and Lending over Stud_no.
   - For each Student row, every Lending row will be retrieved and tested (reading each of the 100 Student rows 10,000 times);
   - Every Lending row will match with one Student row, so the number of joined rows in this intermediate relation is 10,000. These have to written back to disk (only 50 tuples can be held in memory - see assumptions).

   So, the number of tuple l/0s occurred in this step is:
   (100*10,000) + 10,000 = 1,010,000

2. Select the result of Step I for just the tuples for book B I.
   - This results reading the 10,000 joined tuples (obtained in step 1) back into memory.
   - Then Select produces a relation containing only 50 tuples, which can be kept in memory (see assumption).

   The number of tuple l/0s in this step is:
   10,000 + 0 = 10,000

3. Project the result of Step 2 over Stud_Name to get result (50 max).
   - This results in reading a relation of 50 tuples (obtained in step 2) which is already in memory, and producing a final relation of no more than 50 tuples, again in memory.

   The number of tuple I/O in this step is:
   0+0=0

   Therefore, the total number of tuple l/0s for query plan A is:
   (1,010,000 +10,000).

   Total tuple I/O: 1,020,000

Query Execution Plan B - With Optimization

In this approach, the sequence of the operations has been changed to the following:

Select-Join-Project

1. Select the relation Lending for just the tuples for Book B I.
   - This results in reading 10,000 tuples of Lending relation, but only generates a relation with 50 tuples, which will be kept in memory (see assumption).

     The number of tuple I/0s: 10,000 + 0 = 10,000

2. Join the result of Step I with relation Student over Stud_No.
   - This results in reading 100 tuples of Student relation, and joining them with the relation obtained in step I which is already in memory. This join produces a relation of 50 tuples, which again will be kept in memory.

     The number of tuple I/0s: 100 + 0 = 100

3. Project the result of Step 2 over Stud_Name.
   - Same as step 3 of Query Plan A.

     Therefore, the total number of tuple I/0s for query plan B is (10,000 + 100)

     Total tuple I/O: 10,100

Transformation Rules of Query Optimization

There are eight rules in query optimization. These rules are important in second step in the four stages of query processing.

1. Rule I Transform a sequence of restrictions against a given relation into a single (ANDed) restriction.
   (A where Restrict-}) where Restrict-2 = A (where Restrict-I AND Restrict-2)

2. Rule 2 Transform a restriction of a projection into a projection of a restriction.
   A ([Project]) where Restrict = (A where Restrict) [Project]

3. Rule 3 Transform a sequence of projections against a given relation into a single (the last) projection.
   (A [Project-1]) [project-2] s A [Project-2]

4. Rule 4 Distributivity (for restrictions and projections).
   (A Join B) where Restrict-on-A AND Restrict-on-B = (A where Restrict-on-A) Join (B where Restrict-on-B)

5. Rule 5 Distributivity (for logical expressions).
   where p OR (q AND r) = where(pORq)AND(pORr)

6. Rule 6 Choose an optimal ordering of the-joins to keep the intermediate results low in size.
   (A Join B) Join C - A Join (B Join C)

7. Rule 7 Perform projections as early as possible.

8. Rule 8 Perform restrictions as early as possible.

Database Statistics

Various decisions, which have to be made in the optimization process are based upon the database statistics stored in the system, often in the form of a system catalogue or a data dictionary

Typical statistics maintained by the system include:

For each base relation, for example:

- Cardinality of the relation;
- Number of pages for the relation.

For each column of each base relation, for example:

- number of distinct values in this column;
- maximum, minimum, and average value for this column;
- actual values in this column and their frequencies (a-histogram).

For each index, for example:

- whether this is a clustering index;
- number of leaf pages in this index;
- number of levels in this index.

Schedules

Locking schemes can be described as pessimistic, inasmuch as they make the worst-case assumption that every piece of data accessed by a given transaction might be needed by some concurrent transaction and had therefore better be locked.

By contrast, optimistic scheduling also known as certification or validation schemes make the opposite assumption that conflicts are likely to be quite rare in practice. Thus, they operate by allowing transactions to run to completion completely unhindered, and then checking at COMMIT time to see whether a conflict did in fact occur. If it did, the offending transaction is simply started again from the beginning. No updates are ever written to the database prior to successful completion of commit processing, so such restarts do not require any updates to be undone.

Optimistic method have certain inherent advantages over traditional locking methods in terms of the expected level of concurrency (i.e., number of simultaneous transactions) they can support, suggesting that optimistic methods might become the technique of choice in systems with large numbers of parallel processors.

Recoverability

Recoverability means possibility of recovering the database itself in the case of database failure. Therefore Recovery is to "restore the database to a state that is know to be correct after some failure has rendered the current state incorrect, or at least suspect."

The underlying principles for handling recovery can be summarized in a single word “redundancy”. By applying this principle, any piece of information the database contains can be reconstructed (recovered) from some other information stored, redundantly, somewhere else in the system.

If there is a failure in the database Recovery manager in the DBMS will recover the database by using some recovery technique. That technique can be Rollback entire Transaction, Undo or Redo the transactions.

There can be failures due to mainly three reasons.

- Transaction failure
- Media failure
- System failure

To recover from the failure recovery manager will keep track of operations information.

Serialisability of schedules

We have now laid the groundwork for explaining the crucial notion of serializability. Serializability is the generally accepted criterion for correctness for the execution of a given set of transactions. More precisely, a given execution of a set of transactions is considered to be correct if it is serializable—i.e., if it produces the same result as some serial execution of the same transactions, running them one at a time. Here is the justification for this claim:

Individual transactions are assumed to be correct—i.e., they are assumed to transform a correct state of the database into another correct state.

Running the transactions one at a time in any serial order is therefore also correct "any" serial order because individual transactions are assumed to be independent of one another.

An interleaved execution is therefore correct if it is equivalent to some serial execution i.e., if it is serializable.

Terminology: Given a set of transactions, any execution of those transactions, interleaved or otherwise, is called a schedule. Executing the transactions one at a time, with no interleaving, constitutes a serial schedule; a schedule that is not serial is an interleaved schedule (or simply a nonserial schedule). Two schedules are said to be equivalent if they are guaranteed to produce the same result, independent of the initial state of the database. Thus, a schedule is correct (i.e., serializable) if it is equivalent to some serial schedule.

The point is worth emphasizing that two different serial schedules involving the same set of transactions might well produce different results, and hence that two different interleaved schedules involving those transactions might also produce different results and yet both be considered correct. For example, suppose transaction A is of the form "Add 1 to x" and transaction B is of the form "Double x" (where x is some item in the database). Suppose also that the initial value of x is 10. Then the serial schedule A-then-fl gives x = 22, whereas the serial schedule B-then-A gives x = 21. These two results are equally correct, and any schedule that is guaranteed to be equivalent to either A-then-B or B-then-A is like wise correct.

The concept of serializability was first introduced (although not by that name) by Eswaran et al. and he introduce the two-phase locking method.

The two-phase locking method, is as follows:

1. Before start the transaction on any data item (e.g., a database tuple), a transaction must acquire all necessary locks on that data item.
2. After releasing a lock, a transaction must never go on to acquire any more locks.

CHAPTER 7

Concurrency Control Techniques

Introduction

We discussed integrity of the database. The objectives of maintain database integrity is to make sure that the database is an accurate reflection of the real world it is attempting to represent. This is more critical issue in the multi user environment. The kernel of a DBMS is concerning this integrity in multi user environment. Database integrity can be losing due to many reasons.

In this chapter we examine one of the most important aspects of database integrity. By discuss the concurrency problems and possible solutions for them such as locking, time stamping etc.

What is concurrency?

Database systems typically provide multi user access to the database in order to share the data in the database. The process, which enables this simultaneous access to a database, is known as concurrency.

What is Transaction?

In a multi-user database system the events that cause changes to a database or retrieve data from the database are called transactions. A transaction may define as a logical unit of work. It normally corresponds to some coherent activities performed by an individual, group, organization or software process on a database. In practice this can constitute an entire program or a set of commands, which accesses or updates some database.

Concurrency Problems

There are three concurrency problems, i.e. three types of potential mistake which could occur if concurrency control is not properly enforced in the database system.

The lost update problem- This relates to a situation where two concurrent transactions, say A and B, are allowed to update an uncommitted change on the same data item, say x. The second update by transaction B replaces the value of the first update by transaction A. Consequently, the updated value of x by A is lost following the second update by B.

The uncommitted dependency problem- This problem relates to a situation where one transaction A is allowed to retrieve, or update, a data item which has been updated by another transaction B but has not yet been committed by B. Therefore, there is a risk that it may never be committed but be rolled back instead. In that situation, transaction A will have used some data which are now non-existent.

There are two cases in this category:

1. Transaction A uses an uncommitted update by transaction B which is subsequently undone.
2. Transaction A updates an uncommitted change by B whose subsequent rollback loses both of the previous updates.

The inconsistent analysis problem- This problem relates to a situation where transaction A uses an data item which is in an inconsistent state and as a result carries out an inconsistent analysis.

Concurrency Control Techniques

To avoid the above problems there is various solutions. Those are:

Serialization- One solution is to adopt a policy, which permits serial execution of transactions only, i.e. transaction A must process a complete transaction before B can start, or vice versa. The downside of this solution is the slow response time and long CPU idle time.

Concurrency control- Obviously some form of concurrency control mechanism is necessary to enable transactions to run concurrently as far as possible; but controlled in such a way that the effect is the same as if they had been run serially. There are three mechanisms available:

Locking;

- Optimistic scheduling;
- Time stamping.

Locking is the most common mechanism of currency control, therefore is concentrated on in greater detail in this session. It is recommended, however, that an introduction to the basic principle and technique of time stamping is given here.

Locking Techniques for concurrency control

The idea of locking is very simple, when a transaction requires to update a data item in database it require to acquires a lock on that data item. Then until that transaction release the lock that acquires over the data item the data item can't change by the other transactions.

There are main two types of locks:

1. Read lock or Shred Lock (S lock). A read lock gives only read access to data and prevents any other transaction from updating the locked data. Any number of transactions may hold a read lock on a data item and thus they are sometimes referred to as a shared lock. A transaction applies this type of lock when it wishes to query a file but does not want to change it. Also, it does not wish other transactions to change it while it is looking at it.
2. Write locks or Exclusive lock (X lock)**.** A write lock gives both read and writes access to a data item. It also prevents any other transaction from reading from or writing to a data item. For this reason it is often referred to as an exclusive lock.

Locks are normally used in the following way:

1. Any transaction that wishes to access a data item must first lock the data item. If the access is merely read only a read lock is requested. If the transaction wishes to both read and update the data item then a write lock is requested.
2. The DBMS first checks to see if the data item is locked by another transaction. For this purpose the DBMS needs to maintain a lock table. If it is locked, the DBMS determines whether the lock requested is compatible with that set on the data item. If a read lock is requested on a data item already having a read lock on it, then the request is allowed. In all other cases, the requesting transaction must wait.
3. If, of course, the data item is not currently locked then the DBMS allows access and puts a lock against it.
4. A transaction continues to hold a lock until either it releases the lock during its execution or when the transaction terminates.

	B Request			
A has	X	X N	S N	- Y
	S	N	Y	Y
	-	Y	Y	Y

X – Excusive Lock (Write Lock)
S – Shared Lock (Read Lock)
B Transaction Request
A Has locks

A DBMS normally maintains a locking scheme by updating a lock table held in main memory. Imposing a write lock on a data item can have a serious effect on the amount of concurrency in a database system. The degree of seriousness frequently depends on the granularity of the data item. A transaction can acquire Locks on at the whole file, column (a field), row (A record), tuple (One cell). If the locked data items are large, then the performance of update activity can possibly degrade. If the locked data items are at the level of fields, then the size of the lock table held by the DBMS and the amount of processing conducted on the lock table can degrade performance.

Two-Phase Locking

Using locks by themselves does not guarantee serialisability of transactions. To guarantee such serialisability the DBMS must enforce an additional method known as two-phase locking. In this technique divide the life of every transaction into two phases: a growing phase and the shrinking phase

In growing phase, which it acquires all the locks it needs to perform its work, and a shrinking phase, in which it releases its locks. During the growing phase it is not allowed to release any locks, and during the shrinking phase it is not allowed to acquire any new locks. This does not mean that all locks be acquired simultaneously since a transaction normally will engage in the process of acquiring some locks, conducting some processing, then going on to acquire more locks, and so on. The two-phase locking protocol merely involves enforcing two rules:

1. A transaction must acquire a lock on a data item before performing any processing on that data item.
2. Once a transaction releases a lock it is not permitted to acquire any new locks.

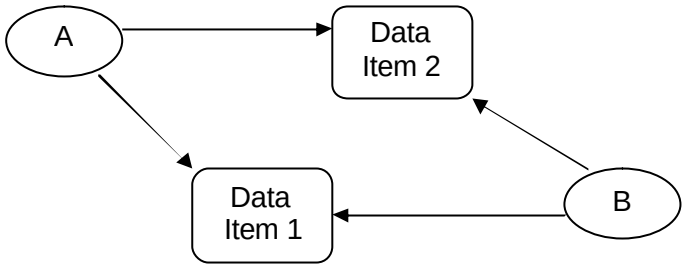
Time stamping and concurrency control.

A data item need not be locked for the entire duration of a transaction. It may simply be locked during data access. This can increase the level of concurrency in a system but increases the likelihood of deadlock occurring. Deadlock, sometimes known as deadly embrace, is best described by a simple example.

Consider two transactions. Transaction A and Transaction B;

- At rime T = 1 Transaction A write locks Data item1.
- At time T = 2 Transaction B write locks Data item2.
- At time T = 3 Transaction A requests a write lock on Data item 2. It must wait since Transaction B has locked Data item 2.
- At time T = 4 Transaction B requests a write lock on Data item1. It must also wait

At this point neither transaction can proceed. This situation is known as deadlock.



There are a number of strategies that can be employed to prevent deadlock. One strategy is to force transaction B to acquire all required locks prior to execution. This strategy, however, tends to have a detrimental effect on performance as large transactions continuously wait to execute.

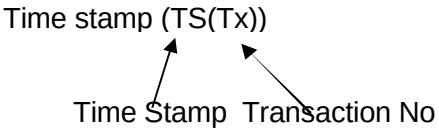
There are more effective strategies in database systems involve detecting deadlock after it has occurred and resolving such deadlock.

A deadlock can detect in the system by searching a cycle the wait-for graph of the database transactions and periodically invokes an algorithm, which searches for a cycle in the graph. Each transaction involved in the cycle is involved in the deadlock.

After detection algorithm has identified a deadlock, the system must try to recover from it. The most common solution is to roll back one or more transactions so that the deadlock can be broken.

Bear in mind that data items held by deadlocked transactions will be unavailable to other transactions until the deadlock is broken.

Determine which transactions, among a set of deadlocked transactions, to roll back to break the deadlock. Transaction can select based on the Time stamping value of the transaction.



Transaction Time stamp is unique identifier assign to the each transaction. The times stamps are ordered based on the order in which transaction are started.

IF T1 start before T2

Then

TS(T1) < TS(T2)

(Older Tx) (Younger Tx)

In the deadlock prevention mechanism system will rollback the youngest transaction out of several transactions in the transactions list.

CHAPTER 8

Data structure for Database Processing

Introduction

This chapter will concentrate on the Data organisation and database processing. Data organisation (sometimes called file organisation) concerns the way in which data is structured on physical storage devices, the most important being disk devices. The idea of data organisation and access is inherently inter-linked. In this chapter we shall discuss the main types of data organisation found in DBMS. We also discuss a type of data organisation known as a cluster, which is particularly commonplace in contemporary relational DBMS.

Data models as described in chapter 2 are all forms of logical data organisation. For instance, relations or tables are logical data structures. A database organized in terms of any particular data model must be mapped onto the organisation relevant to a physical storage device. This form of data organisation is known as physical organisation.

Storage devices and organization

To store the database we can use many data storage devises. In this section we mainly concentrate about disk devices since disks are the most prevalent form of storage device available. The collection of data making up a database must be physically stored on some computer storage medium. Computer storage can be divided into two categories primary storage and secondary storage.

Primary storage includes media that can be directly acted upon by the central processing unit (CPU) of the computer, such as main memory or cache memory. Primary storage usually provides fast access to relatively low volumes of data. Secondary storage cannot be processed directly by the CPU. It hence provides slower access than primary storage but can handle much larger volumes of data. Two of the most popular forms of secondary storage are magnetic disk and magnetic tape.

Most databases are too large to fit in main memory and are therefore usually stored on secondary storage devices such as magnetic disks. Main memory is referred to as volatile memory because the data is lost when the power supply is switched off. Secondary storage is referred to as non-volatile memory because it persists after power loss.

The most basic unit of data stored on disk is the bit (0 or 1). To code information bits are grouped together in bytes or characters, typically 8, 16 or 32 bits making up a byte. The capacity of a disk is hence the number of bytes it can store. Disk capacities are normally described in terms of kilobytes (kbyte: I thousand bytes), megabytes (Mbytes: I million bytes), gigabytes (Gbytes: I billion bytes), or terabytes (Tbytes: I trillion bytes). A typical floppy disk will store from kbytes to Mbytes of data, hard disks for personal computers typically hold from Mbytes to Gbytes of data, and disk packs used in mini and mainframe systems typically store Gbytes to Tbytes of data.

Disks of whatever form are all constructed of thin circles of magnetic material protected by plastic covers. A disk is single-sided if it stores data on only one side of the circle (surface); double-sided otherwise. Disks are assembled into disk packs to increase storage capacity. Disk packs may have as many as 30 surfaces. Data is stored on a disk surface in concentric circles. Each such circle is called a track. In a disk pack, tracks having the same diameter on different surfaces are said to form a cylinder. The number of tracks on a disk typically numbers in the hundreds. Each track stores kbytes of information and is frequently divided into sectors that are hard coded onto the disk. In contrast, a track will frequently be divided up into a number of equal-sized blocks by the operating system during disk formatting. Blocks are hence soft-coded and separated by fixed-size areas known as inter block gaps.

A disk is a random-access device. Data is transferred between main memory and disk in units of blocks (sometimes called pages). The hardware address of a block is composed of a combination of surface number, track number within surface, and block number within track. This address is given to the disk input/output (I/O) hardware. If a read command is received by this hardware it copies the contents of a block to a buffer - a reserved area of main storage that holds one block. For a write command, the hardware copies the contents of the buffer to the disk block.

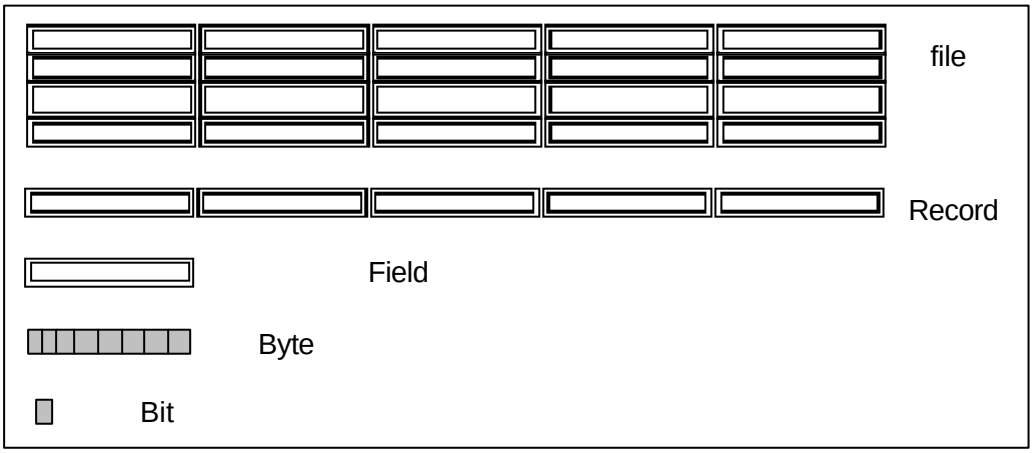


Figure 8.1

Physical representation and Logical record relationships

Data is usually stored physically in the form of records in terms of a given operating system. Each record consists of a collection of related data values or items made up of a number of bytes and corresponding to a particular field. Each record will be of a given record type.

A record type amounts to a format for the records. A file is a collection of such records. Usually all the records in a file will be of the same record type. Figure 8.1 illustrates the hierarchy of physical data structures. Files and records have to be mapped onto blocks, since blocks are the physical unit of disk transfer. Usually, the record size will be less than the block size, meaning that many records will be stored in each block. Consequently a file will be made up of many blocks, frequently but not necessarily contiguous. Each file will normally have a file header. This will contain information used to determine the disk addresses of file blocks and record type information.

Figure 8.2 illustrates the relationship between files and pages/blocks. The module of software concerned with the management of pages on disk is sometimes known as the disk manager. That module of software concerned with the management of files for the operating system is sometimes known as the file manager.

When a file is created, the file manager needs to request a set of blocks from the disk manager. For instance, suppose we work in an environment in which the blocks are fixed at 4 K. A 60 K file will therefore need 15 blocks, and therefore the file manager requests a set of 15 blocks from the disk manager. The set of blocks is given a logical identifier - blocksetID - and each block within the set is assigned a logical ID, usually just its position within the block set.

The file manager knows nothing of the physical storage of data on disk. It merely issues logical block requests (consisting of blocksetIDs and blockIDs) to the disk manager. The disk manager translates the logical block requests from the file manager into physical blockIDs. The disk manager will usually maintain information about the correspondence between logical and physical blockIDs and will contain all the code used to control the disk device. Figure 8.3 illustrates the process by which a DBMS communicates with an operating system's file manager, which in turn issues requests to the disk manager. The DBMS has to translate requests against the constructs of a particular data model (e.g. tables) into requests for files from disk.
The neat separation between the DBMS, file manager and disk manager discussed

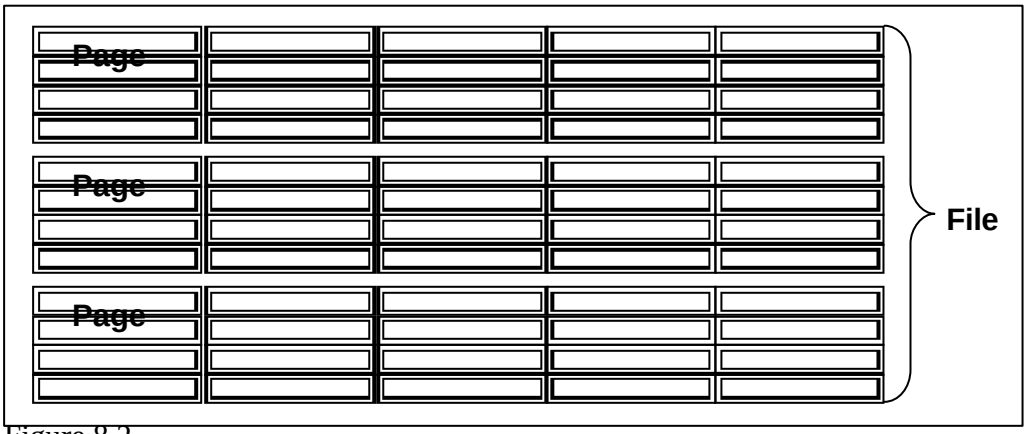


Figure 8.2

above does not always exist in practice. Some file managers are not particularly well suited to the needs of database applications. In this case, the DBMS frequently bypasses the file manager and directly interacts with the disk manager to retrieve data.

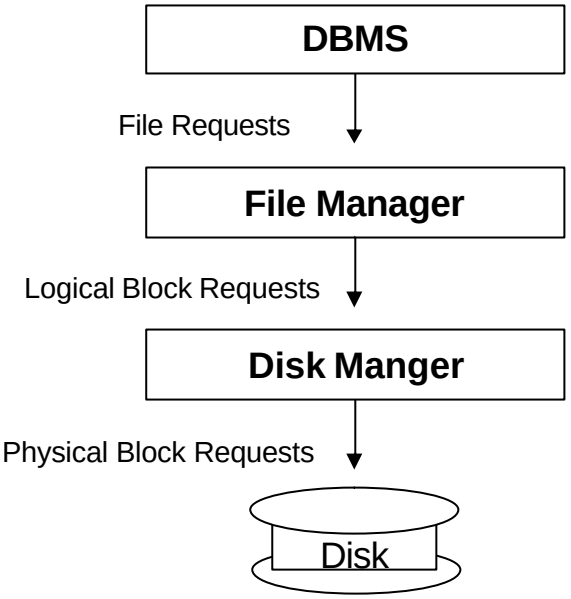


Figure 8.3

File organization method

There 3 main types of file organisation methods:

1. Sequential files - sometime known as serial files, heap files or piles
2. Ordered file
3. Hashed files

Sequential Files

The base form of file organisation is the sequential file. In this form of file organisation, records are placed in the file in the order in which they are inserted. A new record is simply added to the last block in the file. If there is insufficient space in the last block then a new block is added to the file. Therefore, insertion into sequential file is very efficient.

Hence, suppose we have a sequential file of student records. As each student enrolls with the university a new student record is created and added to the end other file. However, searching for a record in a sequential file involves a linear search through the file record by record. Hence, for a file of N records, N/2 records will be searched on average. This makes searching through a sequential file that is more than a few blocks long a slow process.

Another problem arises in relation to deletion activity performed against sequential files. To delete a record we first need to retrieve the appropriate block from disk. The relevant record is then marked as deleted and then written back to disk. Because the deleted records are not reused, a database administrator normally has to re-organise a sequential file periodically to reclaim deleted space.

Ordered Files

The problems of maintenance associated with sequential files mean that most systems tend to maintain some form of ordered file organisation. In an ordered file the records are ordered on the basis of one or more of the fields – generally referred to as the key fields of the file.

If we store student data as an ordered file then we might use student number as our key field.

Ordered files allow more efficient access algorithms to be employed to search a file. One of the most popular of such algorithms is the binary search or binary chop algorithm, which involves a continuous half-wise refinement of the search space.

Insertion and deletion is more complicated in an ordered file. To insert a new record we first have to find the correct position for the insertion in the key sequence. If there is space for the insertion in the relevant block then we can transfer it to main memory, re-order the block by adding the new record, and than write the block back to disk. If there is insufficient space in the block then the record may be written to a temporary area known as overflow. Periodically, when the overflow area becomes full the DBA needs to initiate a merging of the overflow records with the main file.

Hashed Files

Hashed files provide very fast access to records based on certain criteria. They organized in terms of a hash function, which calculates the address of the block record should be assigned to or accessed from.

A hashed file must be declared in terms of a so-called hash key. This means that there can be only one hashed order per file. Inserting a record into a hashed file means that the key of the record is submitted to a hash function. The hash function translates the logical key value into a physical key value - a relative block address. Because this causes records to be randomly distributed throughout a file they are sometimes known as random Files or direct files.

Hash functions are chosen to ensure even distribution of records throughout a file. One technique for constructing a hash function involves applying an arithmetic function to the hash key. For instance, suppose we use student code as the hash key for a student file. The hash function might involve taking the first three characters from a student code, converting these characters to an integer, and adding the result to a similar integer conversion performed on the last three characters of the student code. The result of this addition would then be used as the address of the block into which the record should be placed.

Hashed files must have a mechanism for handling collisions. A collision occurs when two logical values are mapped into the same physical key value. Suppose we have a logical key value: 2034. We find that feeding this key value through the hash function computes the relative block address: 12 (in practice, this address would be a hexadecimal number). A little later we try to insert a record with the key value 5678. We find that this also translates to relative block address 12. This is no problem if there is space in the block for the record. As the size of the file grows, however, blocks are likely to fill up. If the file manager cannot insert a record into the computed block then a collision is said to have occurred. One of the simplest schemes for handling collisions is to declare an overflow area into which collided records are placed. As the hashed file grows, however, the number of records placed in the overflow area increases. This can cause degradation in access performance. For this reason, a number of more complex algorithms are now employed to handle collisions. Various types of hashed file are now also available which dynamically resize themselves in response to update activity.

Clustering

Clustering is a technique whereby the physical organisation of data reflects some aspect of the logical organisation of data. We may distinguish between two types of cluster:

1. Intra-file clustering
2. Inter-file clustering.

Intra-file clustering involves storing data in order of some key value where by each set of records having the same key value are organized as a cluster. For instance, we might decide to cluster a Students table in terms of a department or school code. Hence all computing students would be stored in a cluster, all humanities students would be stored as a cluster, and so on.

Inter-file clustering involves interleaving the data from two or more tables. The table below, for instance, is a clustered version of the Lecturers and Modules tables with which we are familiar.

Cluster: Teaching					
staffNo	staffName	status	moduleName	level	courseCode
234	Davies T	L			
			Relational Database Systems	1	CSD
			Relational Database Design	1	CSD
345	Evans R	PL			
			Deductive Databases	3	CSD
			Object Oriented Databases	3	CSD
237	Jones S	SL			
			Distributed Databases	2	CSD

The rationale for clustering Lecturers data with Modules data in this way is to improve the joining of Lecturer records with Module records. It must be remembered, however, that clustering, like indexing, is a physical concern. All that matters in terms of the data model is that the user perceives the data as being organized in relational terms. How the data is stored on disk is outside the domain of the data model.

The process of creating a cluster will differ among DBMS. However, in general terms, the process will first involve defining the cluster:

CREATE CLUSTER <cluster name>

(<column> <datatype>,...)
(optional sizing information]

For example:
CREATE CLUSTER Teaching
(staffNo NUMBER(5))
We have hence created a cluster named Teaching and declared staffNo to be the so-called cluster key. This will be used to organise the data in the cluster. Next we create the tables and assign each table to the cluster:

CREATE TABLE <table name>
(...)
cluster <duster name> (<table column>)
CREATE TABLE Lecturers
(...)
CLUSTER Teaching (staffNo)
CREATE TABLE Modules
(...)
CLUSTER Teaching (staffNo)

It is valid to use clusters to implement established joins, i.e. stable access paths into your data. However, use of clusters can degrade other access paths, for example, full table scan on single tables in a cluster.

CHAPTER 9

CODASYL Database Model

Introduction

In this chapter we shall examine the essentials of the CODASYL recommendations. We shall not look at any specific CODASYL system. There are many different implementations of the CODASYL recommendations on the market. They all vary in some degree from the CODASYL recommendations, but the variations are for the most part minor ones. And a grasp of the CODASYL specifications is sufficient for an understanding of any of the current implementations. The CODASYL recommendations have developed over a period of about 15 years.

CODASYL is an acronym for Conference on Data System Languages, which is a voluntary organization representing user groups and manufacturers of computer equipment. It was CODASYL that was originally responsible for the development of COBOL.

Architecture of CODASYL Model

The CODASYL approach is embodied in standards recommendations from the COSASYL organization, a voluntary body that was originally responsible for the development of the COBOL programming language. The current CODASYL recommendations have evolved from earlier recommendations developed over the past 15 years, and the latest recommendations closely follow the overall three-level architecture of the ANSI/SPARC data base system proposals: that is we have a conceptual schema, an internal or storage schema, and various external schemas.

As with the hierarchical approach, the conceptual records of a CODASYL database are grouped in two distinct ways:

1. Into groups of conceptual records of the same type, that is, into distinct conceptual files.

2. Into groups of records called owner coupled set types or CODASYL sets. An owner-coupled set grouping of conceptual records embodies a in relationship between conceptual files.

It is worthwhile gaining a grasp of the owner-coupled set concept even at this stage.

Data Definition and Data Manipulation Models

A lot of data base manipulation, and particularly if updating is involved, is done by means of application programs written in host languages such as COBOL or PL/1. Embedded in such application programs are the CODASYL data manipulation language (DML) commands that call up the services of the data base control system that in turn accesses the individual data base records. In addition to application programs, a user group must obtain a definition of that portion of the database it needs to use, that is, it must obtain an external schema.

CODASYL external schema

To a large extent an external schema is merely a subset of the conceptual schema reflecting the fact that an external database is subset of the conceptual database.

CODASYL data manipulation language commands

We have no need to give a detailed (description of all the CODASYL commands order to impart to the reader the basic ideas behind manipulating a CODASL database via an application program. Following are the most common and basic commands that used in the CODASYL.

1. Retrieval commands
    a. Conceptual file FETCH commands
    b. Owner-coupled set FETCH commands
2. Updating commands
    a. There is a MODIFY command that can update one or more fields of a record in the database.
    b. There is an ERASE command for deleting a record in the database
    c. There is a DISCONNECT command for disconnecting a record from an owner-coupled set occurrence of which it is a member; the reef remains in the database.
    d. There is a CONNECT command for disconnecting a record.
    e. There is a CONNECT command for connecting record
    f. There is the STORE command

CODASYL Database Design and Schema and Subschema Descriptions

To a large extent an external schema is merely a subset of the conceptual schema reflecting the fact that an external database is subset of the conceptual database. Note that the CODASYL refers to an external schema as a subschema. Thus an external schema is usually straightforward, and as an example we give an external schema for the portion of the database that is shown in following figure 9.1

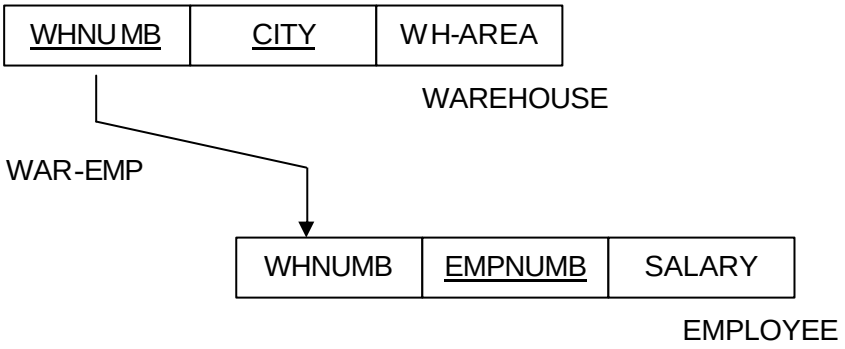


Figure 9.1

The following subschema is for COBOL programs.

```
01      TITLE DIVISION
02      SS PERSONNEL WITHIN OUTSTANDING-ORDERS
03      STRUCTURE DIVISION
04      RECORD      ECTION
05      01      WAREHUSE
06              02      .WHNUMB     ;       PIC X(5)
07              02      .CITY       ;       PIC X(20)
08              02      .WH-AREA    ;       PIC X(6)
09      01      EMPLOYEE
10              02      .WHNUMB     ;       PIC X(5)
11              02      .EMPNUMB    ;       PIC X(5)
12              02      .SALARY     ;       PIC X(6)
13      SET     ECTION
14      SD      AR-EMP
```

A subschema definition language depends on the programming language used with the application program. For example, if we intended to manipulate the data base by means of FORTRAN programs, we would use a subschema definition that was essentially the same but which has a s yntax more like that of FORTRAN. It is quite clear that the subschema above is in the style of COBOL.

The above subschema is called PERSONNEL and is derived from the conceptual schema OUTSTANDING-Of ORDERS.

The subschema also use to define the User Work Area (UWA), which consist of record structure variables each which can hold a record correspond to the external files declaration in the subschema.

CHAPTER 10

Database Administration

Introduction

The database administrator (DBA) is responsible for the technical implementation of database systems, managing the database systems currently in use and setting and enforcing policies for their use. Whereas the data administrator works primarily at the conceptual level of business data, the database administrator works primarily at the physical level. The place where the data administrator and the database administrator meet is at the logical level. Both the data administrator and database administrator must be involved in the system-independent specification and management of data.

The need for a specialist DBA function varies depending on the size of the database system being administered. In terms of a small desktop database system the main user of the database will probably perform all DBA tasks such as taking regular backups of data. However, when many users are using a database and the volume of data is significant, the need for a person or persons, which specializes in administration functions, becomes apparent.

Role of database administrator

The DBA would normally be expected to engage in the following key activities in relation to administering a database.

ADMINISTRATION OF THE DATABASE

1. Physical design. Whereas the data administrator will be concerned with the conceptual and logical design of database systems, the database administrator will be concerned with the physical design and implementation of databases.
2. Data standards and documentation. Ensuring that physical data is documented in a standard way such that multiple applications and end-users can access the data effectively.
3. Monitoring data usage and tuning database structures. Monitoring live running against a database and modifying the schema or access mechanisms to increase the performance of such systems.
4. Data archiving. Establishing a strategy for archiving of 'dead' data.
5. Data backup and recovery. Establishing a procedure for backing-up data and recovering data in the event of hardware or software failure.

ADMINISTRATION OF THE DBMS

The DBA would normally be expected to engage in the following key activities in relation to administering a DBMS:

1. Installation. Taking key responsibility for installing DBMS or DBMS components.
2. Configuration control. Enforcing policies and procedures for managing updates and changes to the software of the database system.
3. Monitoring DBMS usage and tuning DBMS. Monitoring live Tuning of database systems and tailoring elements of the DBMS structure to ensure the effective performance of such systems.

ADMINISTRATION OF THE DATABASE ENVIRONMENT

By administering the database environment we mean monitoring and controlling the access to the database and DBMS by users and application systems. Activities in this area include:

1. Data control. Establishing user groups, assigning passwords, granting access to DBMS facilities, granting access to databases.
2. Impact assessment. Assessing the impact of any changes in the use of data held within database systems.
3. Privacy, security and integrity. Ensuring that the strategies laid down by data administration for data integrity, security and privacy are adhered to at the physical level.
4. Training. Holding responsibility for the education and training of users in the principles and policies of database use.

Privacy and Integrity issues

USERS, USER NAMES AND PASSWORDS

To manage the privacy of the user data DBA define a user as some person authorized to access the database via the DBMS. In most DBMS, a user is known to the database system by his or her user name. This is a string of characters, which the person must type in response to a logon prompt.

Associated with a given user name we usually find a password. This is another string of characters, which must be typed by the person wishing to gain access. Unlike a user name, however, a password being entered is not usually echoed back to the user's screen. The other major difference between user names and passwords is that passwords are normally under the control of a given user. The user can change them at any time. User names, however, are normally created by the DBA and can only be modified by the DBA.

ENFORCING SECURITY AND INTEGRITY

Data integrity involves protecting the database from authorized users of the database. In contrast, data security involves protecting the database from unauthorized users of the database.

Security is normally enforced in a database system using the facilities of user enrolment and assigning privileges to defined users. Some DBMS now enable data to be encrypted particularly in transmission. Encryption involves encoding the data using an algorithm that utilizes encryption and decryption keys. Encrypted data cannot be read without access to the associated decryption key.

Integrity is enforced through integrity constraints. The DBMS will normally enable the DBA to report on active constraints, to drop, enable and disable constraints and to monitor the effect of constraints on update performance.

Backup and Recover procedures

A DBMS will normally provide a facility to allow backup copies of either the whole or part of a database to be made at regular intervals. The DBA must also be able to backup the transaction log or journal. Generally the DBA will be able to backup the database and log with or without (Online backup and Offline Backup) having to stop the database system. Typically, the backup copies are written to off-line storage media such as magnetic tape.

Backups of the database (including the data dictionary) and transaction log may be used to restore the database system in the event of disastrous hardware or software failure. A DBMS normally provides recovery facilities for enabling this restoration process. At longer intervals the DBA may need to purge the database of unused data. Such data may be archived for historical purposes.

Database Access Control

One of the primary functions of the DBA is data control. In any multi-user system, some person or persons have to be given responsibility for allocating resources to the community of users and monitoring the usage of such resources. In database systems, two resources are of pre-eminent importance, Data and DBMS functions.

Data control is the activity, which concerns itself with allocating access to data and allocating access to facilities for manipulating data. Data control is normally the responsibility of the database administrator. In this section we shall examine some of the fundamental aspects of data control in relational database systems. DBA can perform this function with the use of Views.

VIEWS

A view is a virtual table. It has no concrete reality in the sense that no data is actually stored in a view. A view is simply a window into a database in that it presents to the user a particular perspective on the world represented by the database.

We can identify three main interdependent uses for views in a database system: to simplify, to perform some regular function, or to provide security.

In a university we might want to restrict each department or school to only be able to update the data relating to its own students. A view declared on each department or school and attached to a range of user privileges can ensure this. Hence the two views above. Computing Students and Business Students might be secured for access only by administrative staff of the school of computing and the business school, respectively.

Encryption

Encryption is very important issue in the data security. When a user attempts to bypass the system (e.g., by physically removing part of the database, or by tapping into a communication line) the most effective countermeasure against such threats is data encryption—that is, storing and transmitting sensitive data in encrypted form.

In encryption process the plaintext data is encrypted by an encryption algorithm. This encryption algorithm will take plaintext as input and an encryption key; the output from this algorithm is the encrypted form of the plaintext is called the ciphertext. To view the plain text back user need the decryption key. The encryption key is kept secret. The ciphertext, which should be unintelligible to anyone not holding the encryption key, is what is stored in the database or transmitted down the communication line.

CHAPTER 11

Distributed Database

Introduction

The 1990s have been portrayed by many as the era of the distributed database system (Ozsu and Valduriez, 1991). We will describe some of the major features of distributed systems within this chapter. This leads us to a discussion of some of the major types of distributed database system.

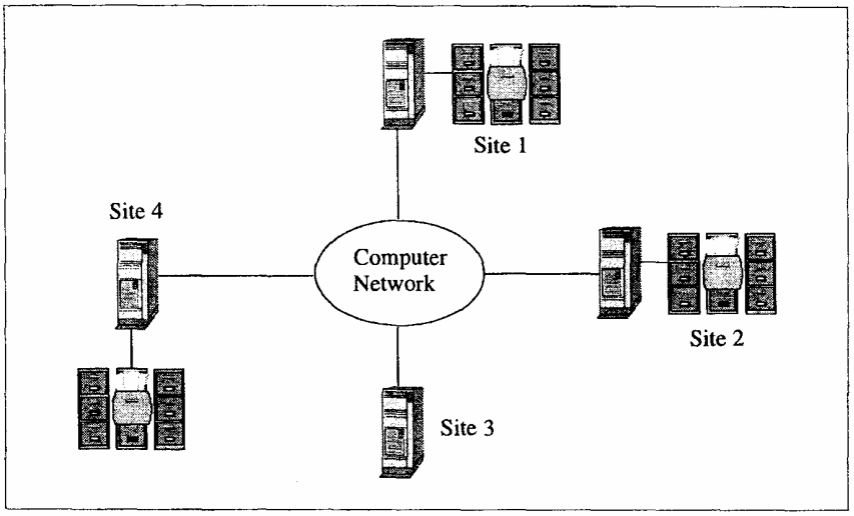
A distributed database system is a database system, which is fragmented or replicated on the various configurations of hardware and software located usually at different geographical sites within an organisation. Distribution is normally discussed solely in terms of the fragmentation and replication of data. A data fragment constitutes some subset of the original database. A data replicate constitutes some copy of the whole or part of the original database. However, distribution can also be discussed in terms of distribution of functions. It is for this reason that we include client-server database systems within our discussion of distributed database systems. Although most current client-server database systems do not effectively distribute data, they do distribute functionality. There for we can define distributed databases as collection multiple, logically interrelated databases, which are physically distributed over a computer network.

According to above definition the words “logically interrelated” are highlighted to emphasize the fact that it only makes sense to group together data in a distributed fashion if there is some relationship between them.

The words “physically distributed” are highlighted to emphasize the fact that two or more separate sites are usual. This will of course have advantages and disadvantages.

The words computer network is highlighted to emphasize the fact that obviously some type of network technology will have to be utilised in this area. Again there will be advantages and disadvantages to this.

"A distributed database management system is defined as the software system that permits the management of the DDBs and makes the distribution transparent to the users." This is simply to clarify that even in the distributed environment a DBMS is necessary for management purposes. It shares many of the aspects of a centralized DBMS, but has to have additional facilities such as making the use of the DDB transparent to the users.



Component of Distributed Database

In distributed databases there are mainly four components. Those are local data processor - The local data processor software is responsible for local data management at a site, much like centralized DBMS software.

1. Data dictionary - The data dictionary holds information on sites, fragments and replicated copies.
2. Remote data processor - The remote data processor software is responsible for most of the distribution functions; it accesses data distribution information from the data dictionary and is responsible for processing all requests that require access to more than one site. An important function of the RDP is to hide the details of data distribution from the user – this is known as Transparency.
3. Network processor - The network processor provides the communication primitives that are used by the RDP to transmit commands and data among the various sites as needed.

Transparency in Distributed Database

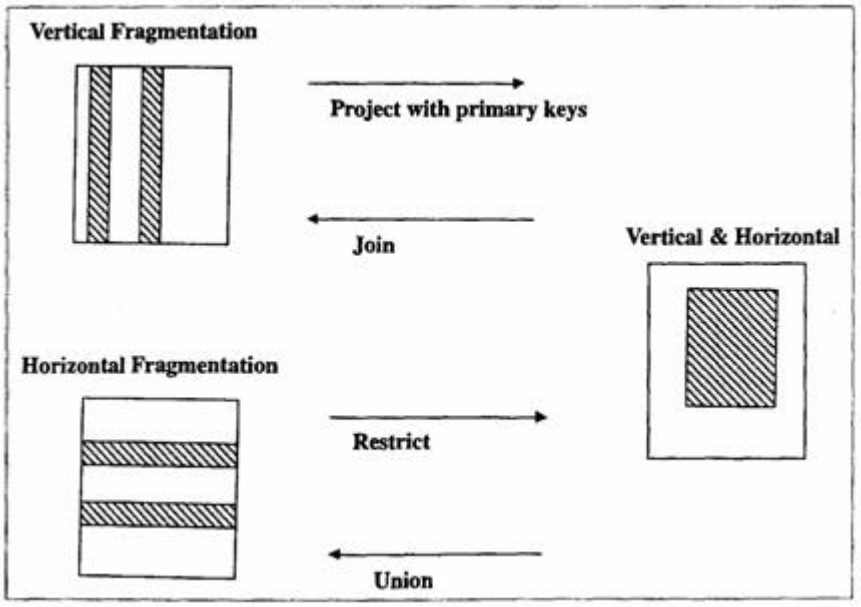
In distributed databases there are three main transparency types. Those are Distribution Transparency - The user should write data access queries and transactions as though the database were centralized, without having to specify the sites at which the data referenced in the query or transaction resides. This property is called distribution transparency.

Replication Transparency - Assuming that data is replicated, the issue related to transparency that needs to be addressed is whether the users should be aware of the existence of copies or whether the system should handle the management of copies and the user should act as if there is a single copy of the data (note that we are not referring to the placement of copies, only their existence).

Fragmentation Transparency - When database objects are fragmented, we have to deal with the problem of handling user queries that were specified on entire relations but now have to be performed on sub relations. In other words, the issue is one of finding a query processing strategy based on the fragments rather than the relations, even though the queries are specified on the latter.

Fragmentation in Distributed Database

Before we decide how to distribute the data, we must determine the logical units of the database that are to be distributed. The simplest of these are the relations themselves. However, in many cases a relation can be divided into smaller logical units for distribution. To do this we need to partition each relation using a technique called fragmentation.



There are three-fragmentation types:

1. Horizontal fragment - In horizontal fragmentation divides a relation horizontally by grouping rows or creating subsets of tuples, where each subset has a certain logical meaning. These fragments can then be assigned to different sites in the distributed system.
2. Vertical fragmentation – In vertical fragmentation a relation is splitting to many relations by keeping only certain attributes in the relation. Those new sub relations are distributing to different sites.
3. Mix fragmentation - We can intermix the above two types of fragmentation, yielding a mixed fragmentation.

Mini Distributed Database

Data mart or mini-warehouse

A "data mart" is a mini-warehouse - typically a Decision Support System for one aspect or branch of a company, with lots of relatively homogeneous data.

Micro Distributed Database

This currently represents the low expenditure end of the distributed database market. However, although a popular term, there is no consensus about what is meant by micro databases. In practice, a micro distributed database system generally refers to a local area network of personal computers (PCs). At least one of these PCs is dedicated to serve the database needs of the others, which act in a client capacity. The database is held on the server. The user interface and application development tools are held on the client machines.

The server in this configuration is either set up as a file server or SQL server. In a file server situation an SQL query expressed by a client will issue a request to the server for the appropriate files needed by the query. The client will perform the query and extract the relevant data. In an SQL server situation the SQL statement will travel down the communication line from client to database server. The server then executes the query and sends back only the extracted data. Clearly, because of the reduced communication traffic, most DBMS now offer SQL server facilities.

Very Large Distributed Database

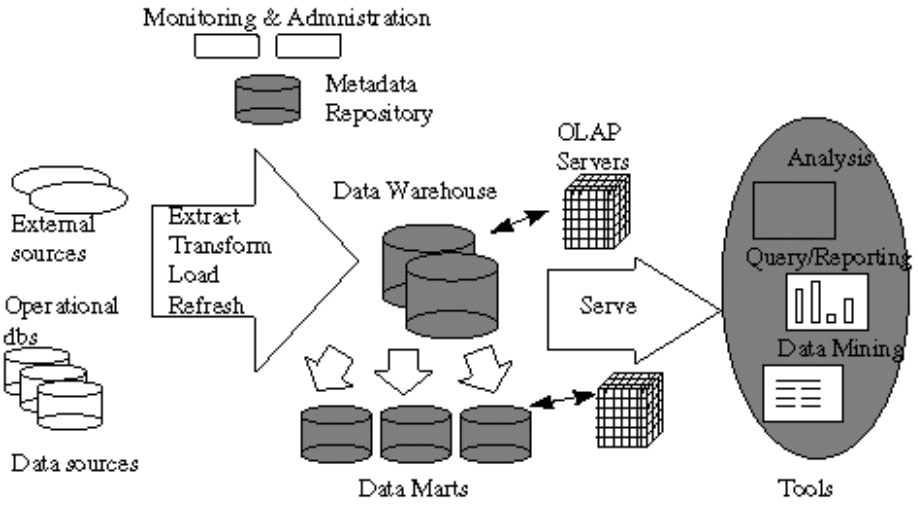
Data warehouse

A "data warehouse" is an organization-wide snapshot of data, typically used for decision-making.

Warehouse properties

- Very large: 100gigabytes to many terabytes (or as big as you can go)
- Tends to include historical data
- Workload: A mostly complex query that access lots of data, and do many scans, joins, aggregations. Tend to look for "the big picture". Some workloads are canned queries (OLAP), some are ad-hoc (general DSS). Parallelism is must.
- Updates pumped to warehouse in batches (overnight)
- Data may be heavily summarized and/or consolidated in advance (must be done in batches too, must finish overnight). This is where the lion's share of the research work has been (e.g. "materialized views") -- a small piece of the problem.

A typical data warehousing architecture (c. 1996):



Data Cleaning

- Data Migration: simple transformation rules (replace "gender" with "sex")
- Data Scrubbing: use domain-specific knowledge (e.g. zip codes) to modify data. Try parsing and fuzzy matching from multiple sources.
- Data Auditing: discover rules and relationships (or signal violations thereof). Not unlike data "mining".

Data Load: can take a very long time! (Sorting, indexing, summarization, integrity constraint checking, etc.) Parallelism a must.

- Full load: like one big exact – change from old data to new is atomic.
- Incremental loading ("refresh") makes sense for big warehouses, but transaction model is more complex – have to break the load into lots of transactions, and commit them periodically to avoid locking everything. Need to be careful to keep metadata & indices consistent along the way.

Design of Distributed Database System

The ability to distribute data among different nodes in a network is now a commonplace feature of modern DBMS. The design for distributed database systems is therefore an important aspect of modern database design. Distributed database design can be seen as a variant of physical database design.

The first stage in distributed database design is to develop a data model in the same way as for a centralized system. The requirements of distribution may then mean that some changes need to be made to the data model.

The second stage is to decide how to fragment and replicate the logical schema. A data fragment represents a horizontal and/or vertical fragment of tables in the database. The main aim of distributed database design is to reduce communications traffic. We wish to locate data close to its point of use. Hence a consideration of the topology of the communications network will influence the distributed design.

SKILLS DEVELOPMENT PROJECT
Ministry of Tertiary Education & Training

National Diploma in Information & Communication Technology

Database Management System

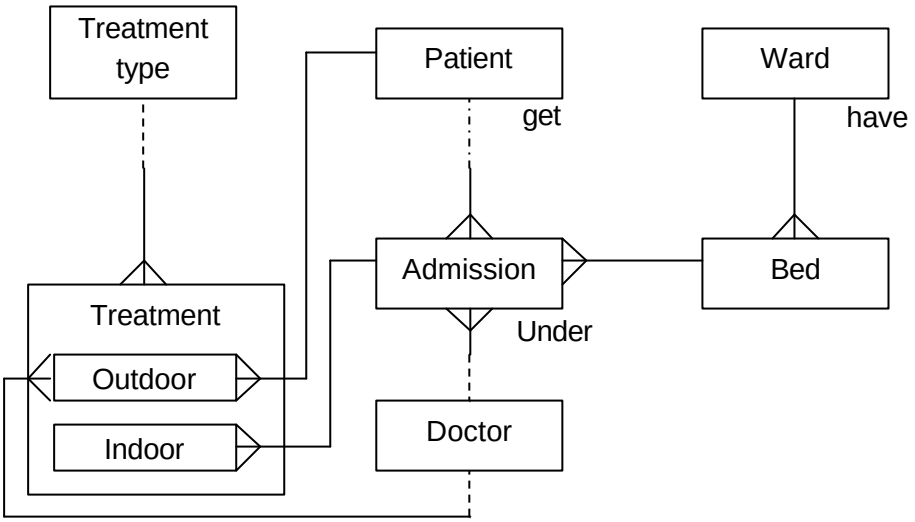
Assignments

209

Developed by
Interactive Training Division
IDM Computer Studies (Pvt) Ltd.
http://www.idm.edu

ASSIGNMENT 1

Hospital medical system



This is ERD for the medical system database of the small hospital. Following assignments are based on this ERD.

Assignments

Question 01

It is required to implement the above ERD as a Relational Database. Write down all the corresponding relational tables by clearly indicating their primary keys and foreign Keys. The non-key attributes are not required.

Question 02

Define the following with reference to Relational Database concepts by giving example from above system.

- Candidate Key
- Primary Key
- Foreign Key

Question 03

"How data redundancy could affect data inconsistency". Explain with an example from above system.

Question 04

What is data redundancy and how it has been avoided in Relational databases?

Question 05

Discuss the advantages of having a database over a traditional filling system.

Question 06

Write down the following rules and discuss their applications in above Relational Database.

- Entity Integrity Rule
- Referential Integrity Rule

ASSIGNMENT 2

Research and Development (R&D) company system

A Research and Development (R&D) company employs a number of scientists who are working a number of projects in different locations in Sri Lanka. Each project will involve several scientists who may need some research facilities, to purchase materials, to go on field trips, etc. Some projects may produce patents or royalties. The performance of each scientist is evaluated based on the number of seminar presentations given, publications issued, revenue generated, etc.

As the company grows, it is planning to implement an MIS using a distributed database to manage its operation and, in particular, to monitor projects and their corresponding usage of scientists and resources.

The system should enable the users to;-

- Monitor the involvement of each scientist, the facilities and other resources required, and the revenue generated by individual projects so as to give an estimate of the total project cost.
- Estimate the amount of incentives for each scientist based on their project involvement.
- Search for a scientist with a relevant field of research for a specific project.

Assignments

Question 01

Explain what is meant by DISTRIBUTED DATABASE. Discuss key ADVANTAGES of managing data using a database.

Question 02

Plan and Design suitable fragmentation plan of relations for the distributed database.

Question 03

What is meant by the “transparency” in the distributed databases?

Question 04

What is data redundancy and how it has been avoided in distributed databases?

Question 05

Discuss the advantages of having distributed databases over a traditional centralized database system.

Question 06

Discuss the Role of Database Administrator in the above system.

ASSIGNMENT 3

Library Database System

The management of the small local library decides to make automated computer system to maintain the existing manual procedures of the library. Within this new system they planed several database files to hold the various types of data for process.

Assignments

Question 01

Identify FOUR of the main types of data files used in data processing.

Question 02

Explain how data in the MEMBER table can be stored as an INDEXED SEQUENTIAL file. Use a diagram to illustrate your answer. Assume that about 5 records will fit onto each disc page. Your diagram should show the required index layers as well as the data pages.

Question 03

Explain how overflow can occur in INDEXED SEQUENTIAL structures and describe how it is resolved.

Question 04

A decision has been taken to store the BOOK table as an INDEXED SERIAL file.

Compared to an indexed sequential file, explain why this storage method is less efficient in terms of access to the data.

Question 05

In relational DBMSs 'clusters' may be implemented. Explain how the use of many clusters can degrade overall performance of the database.

ASSIGNMENT 4

Order Processing System

In order processing system there are several data files to hold the various related information's. Followings are some of those files.

Customer, Order and item. The attributes highlighted are the keys for the relevant relations:

Customer = (**Cust_No**, Cust_Name, Gender, Address)

Order = (**Order_No**, Cust_No, Item_No)

Item = (**Item_No**, decs, price, weight)

Consider the following query:

Retrieve the names of customers who have ordered the item no 001.

This query can be expressed in SQL:

Select Distinct Customer.Cust_Name

From Customer, order

Where Customer.C ust_No = Order.Cust_No

And Order.Item_No='001'

We further make the following two assumptions:

The database contains 1000 Customers and 10,000 Orders, of which only 100 and for Item 001.

It is possible to hold up to 50 tuples in memory, without having to write back to disk.

Assignments

Question 01

Write a series of relational algebra statements to process the same query as specified in above

Question 02

Calculate how many tuples are read as each statement is processed

Question 03

Calculate bow many rows and columns there would be in the resulting relation

Question 04

State the total number of tuples read in the processing of the query.

Question 05

Rewrite the same query in order to optimize and Calculate how many tuples are read as each statement is processed

Question 06

Calculate bow many rows and columns there would be in the new resulting relation

ASSIGNMENT 5

TERATOY Database System

"TERATOY" is a retail company that sells children's toys. The IT manager at TERATOY has created a TERATOY database to control stocks of the company. All the staff of the company uses this TERATOY database in multi user environment.

Assignments

Question 01

Describe the Database transaction concept in respect to the multi user environment.

Question 02

Describe THREE well-known problems, which can be caused by interleaving the operations of concurrent transactions.

Question 03

Two main solutions to overcome the problems in (Q2) are locking and time stamping. Explain how locking and time stamping overcome the above problems.

Question 04

What is deadlock? Show, with the aid of a timing diagram, how locking can cause deadlocks.

SKILLS DEVELOPMENT PROJECT
Ministry of Tertiary Education & Training

National Diploma in Information & Communication Technology

Database Management System

Case Studies

209

Developed by
Interactive Training Division
IDM Computer Studies (Pvt) Ltd.
http://www.idm.edu

CASE STUDY 1

Database system for a small local library

There is a small local library, and all the activities are performed manually. The local authority needs to automate existing manual library system. Suitable database is required for that system to provide services for them.

Description

The new system planed to provide Services/functions to Support book loans, book reservations, issue of overdue loan reminders, book registration, and reader registration.

Following services/functions are required by the system:

- To lend a book copy - Loans are one week long. The maximum number of books on loan for each reader is 10. Readers who reach this limit or with outstanding overdue loans can not borrow more books. If the reader borrows the book reserved by him/her, the corresponding reservation is terminated.
- To send reminders - Overdue loans should be monitored and up to two reminders sent. When a reminder is sent its date will be stored in the loan record.
- To reserve a book title - A reader-can reserve a title if all copies are out on loan.
- To deal with a book return - When a book copy is returned by a reader, the corresponding loan record is updated and outstanding reservations are checked. If the book has been reserved, the reservation record is updated to indicate which copy is available and the appropriate reader is notified.
- To register a new reader - This creates the reader's record.
- To deregister an existing reader - This deletes the reader's record.
- To add a new book copy - This creates the book copy record and the book title record
- (if the first copy of the title is added).
- To remove an existing book copy - This deletes the book copy record and the book title record (if the last copy is removed).

To simplify this case study, other aspects of the system (for example, fines) are not specified.

Requirements

1. Describe the database design process of above library system by dividing the process in to three steps:
    a. Conceptual database design
    b. Logical database design
    c. Physical database design

2. Workout the relational data analysis (Normalization) of above system by giving the important steps.

3. Draw the Entity event matrix by giving all the entities and the their corresponding events

4. Draw the Entity relationships diagram with their attributes for the library system to provide the required services

5. Draw the set relational database tables to implement the ER diagram that you create in the above question.

6. Plan the physical database design to implement the above system with specifying the software and hardware requirements.

CASE STUDY 2

Database system for Manufacturing Company

Description

A Manufacturing Company makes Parts and assembles these Parts to make a Product (such as a universal steering joint, a shock absorber). A Part can be used in more than one Product. Each Part is made on a machine (such as a capstan lathe) by a Machinist, an employee of the Company. Machinists work on a particular machine throughout their employment but Parts can be made on different machines as tool sets can be changed for machining a different Part. Skilled employees called Setters supervise and manage a number of machines during the manufacturing process. A customer can buy Products by completing a Job sheet: this gives details such as the quantity of each product requested and the selling price.

Data processing operations required:

- List employees (machinists and setters) involved in the manufacture of Product No 3498 (a back axle).
- What is the total cost of the parts in the assembly of one product with Product No 2209 (a shock absorber)?
- How many Parts has each machinist made today?
- What is the average salary of Setters employed by the company?
- What Parts are used in the assembly of Product No 2209?
- Which machines are located in the Milling Shop?
- What customer has ordered more than 5 products with Product No 2209 on a single order?

State any assumptions you make and include any necessary entities or services.

Requirements

1. Workout the relational data analysis (Normalization) of above system by giving the important steps.

2. Draw the Entity event matrix by giving all the entities and the their corresponding events

3. Draw the Entity relationships diagram with their attributes for the above system to provide the required services

4. Draw the set relational database tables to implement the ER diagram that you create in the above question.

5. What are the data base constraints required to maintain the database integrity? Justify your solutions.

6. Company establishes the branches in different locations with the development of the company. Company management decides to convert existing centralized database system to distributed database. How you can convert the existing database to distributed database. Describe the design process.

7. How you can plan the fragmentations in the new database system? Justify your answer.