1. Database Fundamentals

Abstract: The aim of this course is to familiarize the students with the fundamental concepts of database administration. The course presents the database and data model concepts, as well as the process of developing and planning a database. A particular accent is placed on the relational model as this represents the most used concept in most of the modern-day database systems.

Contents

Objective:

- Introduce the students with the universe of database administration
- Introduce the concepts of database, database management systems and data models
- Describe the relational model and databases
- Familiarize the students with the role and activities of the database administrator

1.1. Database, DBMS and Data Model

We start by explaining simple concepts in database design, such as the difference between a database model and a database. A ***database model*** is a blueprint for how data is stored in a database and is similar to an architectural approach for how data is stored – this is commonly known as an ***entity relationship diagram*** (a database on paper). A ***database*** is the implementation of a physical database model on a computer. So a database model is used to create a database.

A database is a collection of information—preferably related information and preferably organized. A database consists of the physical files you set up on a computer when installing the database software. On the other hand, a database model is more of a concept than a physical object and is used to create the tables in your database.

By definition, a database is a structured object. It can be a pile of papers, but most likely in the modern world it exists on a computer system. That structured object consists of data and metadata, with metadata being the structured part. Data in a database is the actual stored descriptive information, such as all the names and addresses of your customers. Metadata describes the structure applied by the database to the customer data. In other words, the metadata is the customer table definition. The customer table definition contains the fields for

the names and addresses, the lengths of each of those fields, and datatypes. (A datatype restricts values in fields, such as allowing only a date, or a number). Metadata applies structure and organization to raw data.

The term database usually implies a series of related properties:

- *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.

- *Data Integration* – Shared data brings numerous advantages to the organisation. 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. A data value is redundant when an attribute has two or more identical values. A data value is redundant if you can delete it without information being lost.

- *Data Integrity* – Another responsibility arising as a consequence of shared data is that a database should display integrity. In other words, the database should accurately reflect the universe of discourse that it is attempting to model. This means that if relationships exist in the real world between objects represented by data in a database then changes made to one partner in such a relationship should be accurately reflected in changes made to other partners in that relationship.

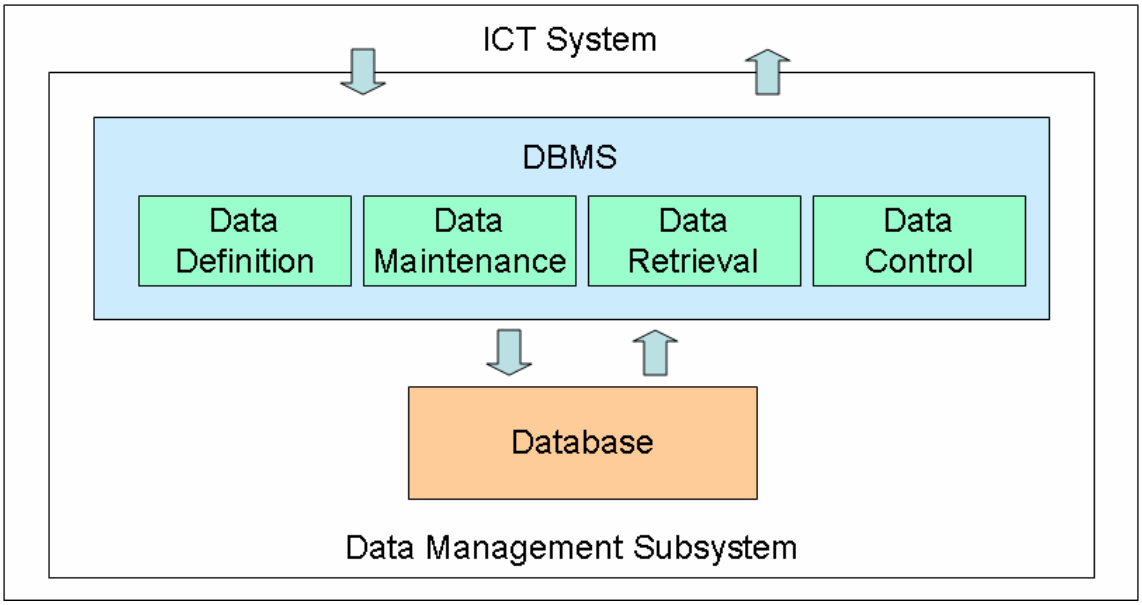
- *Data Security* – One of the major ways of ensuring the integrity of a database is by restricting access – in other words, securing the database. The main way this is done in contemporary database systems is by defining in some detail a set of authorised users of the whole, or more usually parts, of the database.

- *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. Hence, for instance, an academic database is meant to record relevant details of university activity. We say relevant, because no database can store all the properties of real-world objects. A database is therefore an abstraction of 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 organisation 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. Properties such as data independence are only partly achieved in current implementations of database technology.

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 though which all interactions take place with the database. The interactions supplied by most existing DBMS fall into four main groups, as follows:



- *Data definition* – defining new data structures for a database, removing data structures from the database, modifying the structure of existing data.
- *Data maintenance* – inserting new data into existing data structures, updating data in existing data structures, deleting data from existing data structures.
- *Data retrieval* – querying existing data by end-users and extracting data for use by application programs.
- *Data control* – creating and monitoring users of the database, restricting access to data in the database and monitoring the performance of databases.

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

1. Data Model as Architecture

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 the object-oriented data model. This set of principles that define a data model may be divided into three major parts:

- *Data definition* – a set of principles concerned with how data is structured.
- *Data manipulation* – a set of principles concerned with how data is operated upon.
- *Data integrity* – a set of principles concerned with determining which states are valid for a database.

Data definition involves defining an organisation for data: a set of templates for the organisation of data. 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. For example, in the relational data model described in the next section, 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 that are inherently part of the data model. A great deal of other integrity constraints or rules have to be specified by additional means, i.e. using mechanisms not inherently part of the data model.

2. Data Model as Blueprint

The term data model is also used to refer to an integrated, but implementation independent, set of data requirements for some application. We 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.

We can make a distinction between three generations of architectural data model:

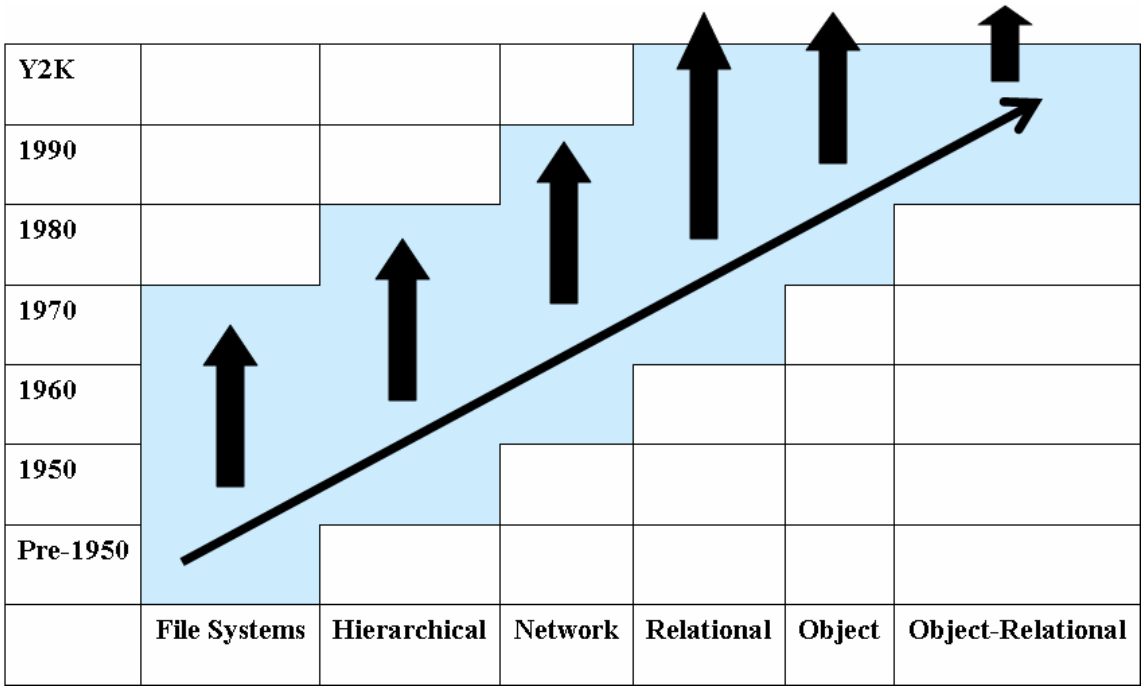
- *Primitive data models*. In this approach objects are represented by record structures grouped in file-structures. The main operations available are read and write operations over records.
- *Classic data models*. These are the hierarchical, network and relational data models. The hierarchical data 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.
- *Semantic data models*. The main problem with classic data models such as 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. Examples of semantic data models are the post-relational data model and the object-oriented data model.

The various data models that came before the relational database model (such as the hierarchical database model and the network database model) were partial solutions to the never-ending problem of how to store data and how to do it efficiently. The relational database model is currently the best solution for both storage and retrieval of data. Examining the relational database model from its roots can help you understand critical problems the relational database model is used to solve; therefore, it is essential that you understand how the different data models evolved into the relational database model as it is today.

The evolution of database modelling occurred when each database model improved upon the previous one. The initial solution was no virtually database model at all: the file system (also known as flat files). The file system is the operating system. You can examine files in the file system of the operating system by running a dir command in DOS, an ls command in UNIX, or searching through the Windows Explorer in Microsoft Windows. The problem that using a file system presents is no database structure at all.

The next figure shows that evolutionary process over time from around the late 1940s through and beyond the turn of the millennium, 50 years later. It is very unlikely that network and hierarchical databases are still in use.



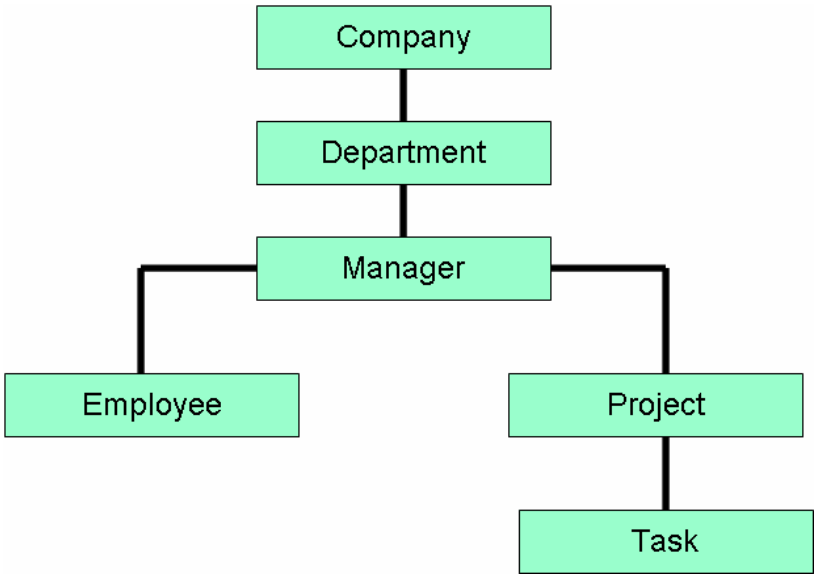
Using a ***file system database model*** implies that no modelling techniques are applied and that the database is stored in flat files in a file system, utilizing the structure of the operating system alone. The term “flat file” is a way of describing a simple text file, containing no structure whatsoever—data is simply dumped in a file.

By definition, a comma-delimited file (CSV file) contains structure because it contains commas. By definition, a comma-delimited file is a flat file. However, flat file databases in the past tended to use huge strings, with no commas and no new lines. Data items were found based on a position in the file. In this respect, a comma-delimited CSV file used with Excel is not a flat file.

Any searching through flat files for data has to be explicitly programmed. The advantage of the various database models is that they provide some of this programming for you. For a file system database, data can be stored in individual files or multiple files. Similar to searching through flat files, any relationships and validation between different flat files would have to be programmed and likely be of limited capability.

The ***hierarchical database model*** is an inverted tree-like structure. The tables of this model take on a child-parent relationship. Each child table has a single parent table, and each parent table can have multiple child tables. Child tables are completely dependent on parent tables; therefore, a child table can exist only if its parent table does. It follows that any entries in child tables can only exist where corresponding parent entries exist in parent tables. The result of this structure is that the hierarchical database model supports one-to-many relationships.

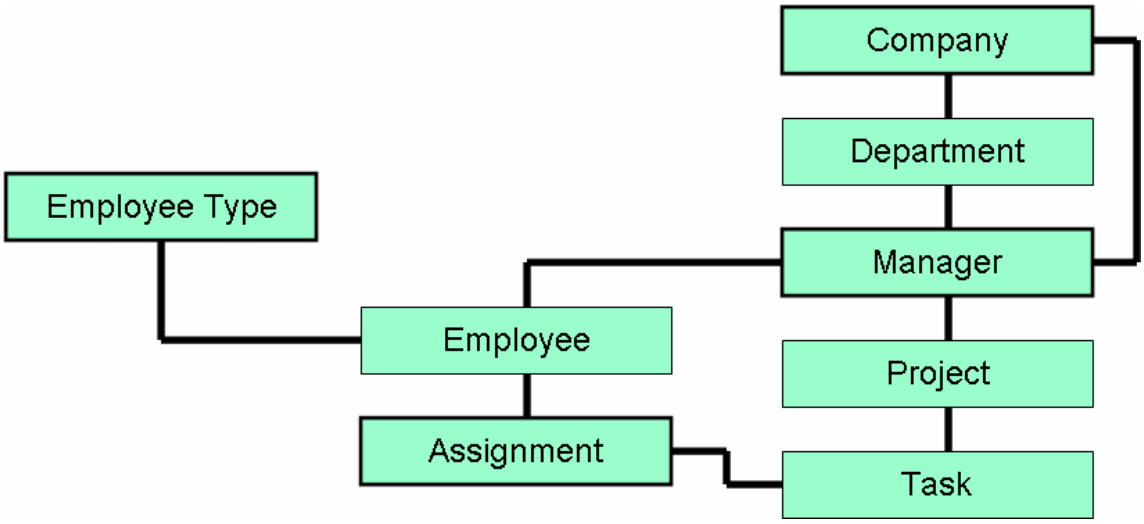
The next figure shows an example hierarchical database model. Every task is part of a project, which is part of a manager, which is part of a division, which is part of a company. So, for example, there is a one-to-many relationship between companies and departments because there are many departments in every company. The disadvantages of the hierarchical database model are that any access must originate at the root node, in the case of the example presented in the figure, the Company. You cannot search for an employee without first finding the company, the department, the employee's manager, and finally the employee.



The ***network database model*** is essentially a refinement of the hierarchical database model. The network model allows child tables to have more than one parent, thus creating a networked-like table structure. Multiple parent tables for each child allows for many-to-many

relationships, in addition to one-to-many relationships. In an example network database model shown in the next figure, there is a many-to-many relationship between employees and tasks. In other words, an employee can be assigned many tasks, and a task can be assigned to many different employees. Thus, many employees have many tasks, and visa versa.

The example shows how the managers can be part of both departments and companies. In other words, the network model in this example is taking into account that not only does each department within a company have a manager, but also that each company has an overall manager (in real life, a Chief Executive Officer, or CEO). The example also shows the addition of table types where employees can be defined as being of different types (such as full-time, part-time, or contract employees). Most importantly to note is the new Assignment table allowing for the assignment of tasks to employees. The creation of the Assignment table is a direct result of the addition of the multiple-parent capability between the hierarchical and network models. As already stated, the relationship between the employee and task tables is a many-to-many relationship, where each employee can be assigned multiple tasks and each task can be assigned to multiple employees. The Assignment table resolves the dilemma of the many-to-many relationship by allowing a unique definition for the combination of employee and task. Without that unique definition, finding a single assignment would be impossible.



The relational database model, which is the most widely used today, is covered in the next section.

The term ***database system*** is used to include the concepts of a data model, DBMS and database. To build a database system we must use the facilities of some DBMS to define suitable data structures – the schema. We also use the DBMS to define integrity constraints. We then populate such data structures using the manipulative facilities of the DBMS. The DBMS will enforce integrity on any manipulative activity.

Both the DBMS and the resulting database must adhere to the principles of some data model. In practice, there is usually some divergence between the abstract ideas specified in a data model and the practical implementation of ideas in a DBMS. This is particularly evident in

the area of relational database systems where the relational data model is not completely represented by any existing DBMS.

A Brief History of Data Management

Database systems are tools for data management and form a branch of information and communications technology. Data management is an ancient activity and the use of ICT for such activity is a very recent phenomenon. Gray (1996) provides a useful categorisation of data management into six historical phases. We have adapted this somewhat to take account of recent developments:

1. MANUAL RECORD MANAGERS (4000 BC–AD 1900)

Human beings have been keeping data records for many thousands of years. Interestingly, some of the earliest recorded uses are for the representation of economic information. For example, the first known writing describes the royal assets and taxes in Sumeria dating to 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 six thousand years.

2. PUNCHED-CARD RECORD MANAGERS (1900–55)

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.

3. PROGRAMMED RECORD MANAGERS (1955–70)

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 transactions 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 run, which typically ran overnight.

4. ON-LINE NETWORK DATA MANAGERS (1965–80)

Applications such as 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 organisations began experimenting with the provision of on-line transaction databases. Such databases became feasible with developments in tele-processing 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 organisation 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. These early on-line systems pioneered solutions to running many concurrent update activities against a database shared among 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 beginning of a company named Cullinane, later Cullinet, which established the IDMS DBMS as the dominant force of network DBMS on IBM mainframes in the 1960s, 1970s and 1980s.

5. RELATIONAL DATA MANAGEMENT (1980–95)

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 an IBM scientist, Dr E.F. Codd, outlined the relational data model which offered higher-level definition and manipulation of data, and 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 gained 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 received 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 computing, 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 has 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. Many modern relational DBMS are offering parallel processing capabilities.

6. MULTIMEDIA DATABASES (1995–99)

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 simply stored data. The processing of data was accomplished by application programs working outside, 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.

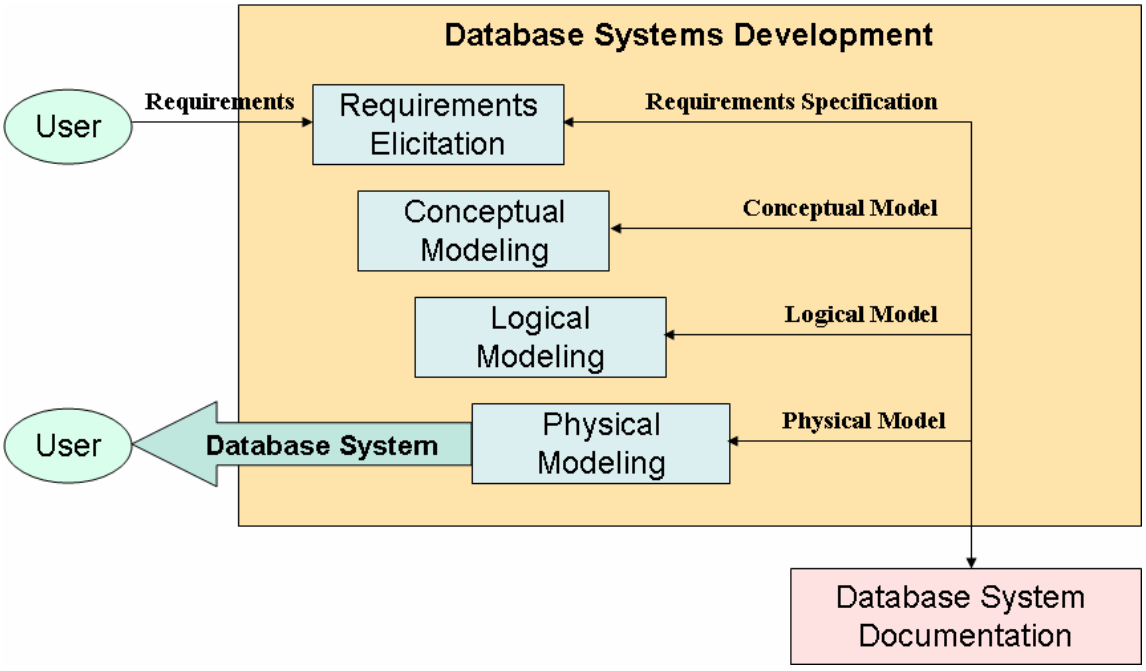
In recent years many claims have been made for new types of database systems based on object-oriented ideas. A number of commercial object-oriented DBMS have now become

available. Many of the relational vendors are also beginning to offer object-oriented features in their products, particularly to enable handling complex data.

7. WEB-BASED SYSTEMS (1999–)

We would add a further phase to the typology outlined above – that of Web-based systems. The World-Wide-Web, or Web for short, is a set of technology standards for managing and disseminating information over the Internet. Web-based standards have become dominant in the definition of front-end ICT systems. Contemporary databases provide the data for such systems and hence a great deal of activity has occurred in suggesting various approaches for integrating DBMS with Web-sites. Certain Web-based standards, such as XML, are also proving significant in attempts to integrate diverse database systems.

Producing a database model is an important part of the process of ***database development***. Database development is generally a process of modeling. It is a process of successive refinement through three levels of model: conceptual models, logical models and physical models. A conceptual model is a model of the real world expressed in terms of data requirements. A logical model is a model of the real world expressed in terms of the principles of some data model. A physical model is a model of the real world expressed in terms of the constructs of some DBMS such as tables, and access structures such as indexes. Hence there are three core stages to any database development task: *conceptual modeling*, *logical modeling* and *physical modeling*:



Some people see ***conceptual modeling*** as a stage headed by a requirements elicitation. This process involves eliciting the initial set of data 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.

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 the next section 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 the Database Administration course we are interested in the use of the databases in the context of the ***ICT (Information and Communications Technology) systems***. Information and Communications Technology is any technology used to support information gathering, processing, distribution and use. ICT provides a means of constructing aspects of information systems, but is distinct from information systems. Modern ICT consists of hardware, software, data and communications technology:

- *Hardware*. This comprises the physical (hard) aspects of ICT consisting of processors, input devices and output devices. Examples include devices such as keyboard (input), processing units and monitors (output).

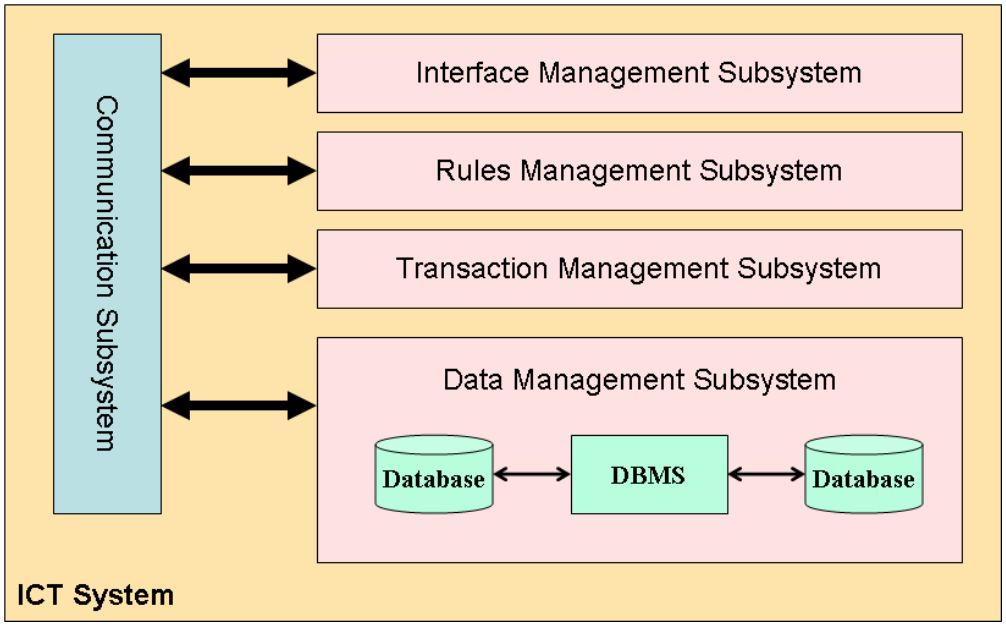
- *Software*. This comprises the non-physical (soft) aspects of information technology. Software is essentially programs – sets of instructions for controlling computer hardware. Examples include operating systems, programming languages and office packages.

- *Data*. This constitutes a series of structures for storing data on peripheral devices such as hard disks. Such data is manipulated by programs and transmitted via communication technology. As example consider the data that is normally stored in databases managed by a database management system.

- *Communication technology*. This forms the interconnectivity tissue of information technology. Communication networks between computing devices are essential elements of the modern ICT infrastructure of organizations. Communication technology includes components such as cabling, transmitters and routers.

An information and communications technology system is a technical system. Such systems are frequently referred to as examples of ‘hard’ systems in the sense that they have a physical existence. An ICT system is an organized collection of hardware, software, data and communications technology designed to support aspects of some information system. An ICT system has data as input, manipulates such data as a process and outputs manipulated data for interpretation within some human activity system.

It is useful to consider an ICT system as being made up of a number of subsystems or layers:



- *Interface subsystem*. This subsystem is responsible for managing all interaction with the end-user. Hence, it is frequently referred to as the user interface.

- *Rules subsystem*. This subsystem manages the application logic in terms of a defined model of business rules.

- *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 initiate actions (responses or changes) in the data subsystem.

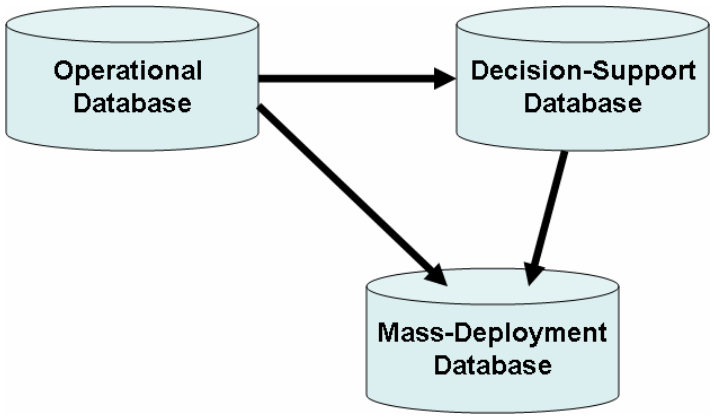
- *Data subsystem*. This subsystem is responsible for managing the underlying data need by an application.

In the contemporary ICT 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. This facility is known as the communications subsystem.

In terms of the data management layer, aspects of the processing of this layer or aspects of the data may be distributed. Historically, the four parts of a conventional ICT application were constructed using one tool, the high-level or third generation programming language (3GL). A language such as COBOL was used to declare appropriate file structures (data subsystem), encode the necessary operations on files (transaction subsystem), validate data processed (rules subsystem) and manage the terminal screen for data entry and retrieval. However, over the last couple of decades there has been a tendency to use a different, specialized tool for one or more of these layers. For instance:

- Graphical user interface tools have been developed as a means of constructing sophisticated user interfaces.
- Fourth generation languages have been developed as a means of coding business rules and application logic.
- Transaction processing systems have been developed to enable high throughput of transactions.
- Database management systems have been developed as sophisticated tools for managing multi-user access to data.
- Communications are enabled by a vast array of software supporting local area and wide area communications.

ICT systems are critical components of contemporary information systems. Database systems are critical elements of ICT systems. In such systems databases serve three primary purposes: as operational tools, as tools for supporting decision-making and as tools for deploying data around the organization:



1. OPERATIONAL DATABASES

Such databases are used to collect operational data. Operational databases are used to support standard organizational functions by providing reliable, timely and valid data. The primary usages of such databases include the creating, reading, updating and deleting of data – sometimes referred to as the CRUD activities. In terms of a university, an operational database will probably be needed to maintain an ongoing record of student progression.

2. 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. In terms of a university, a

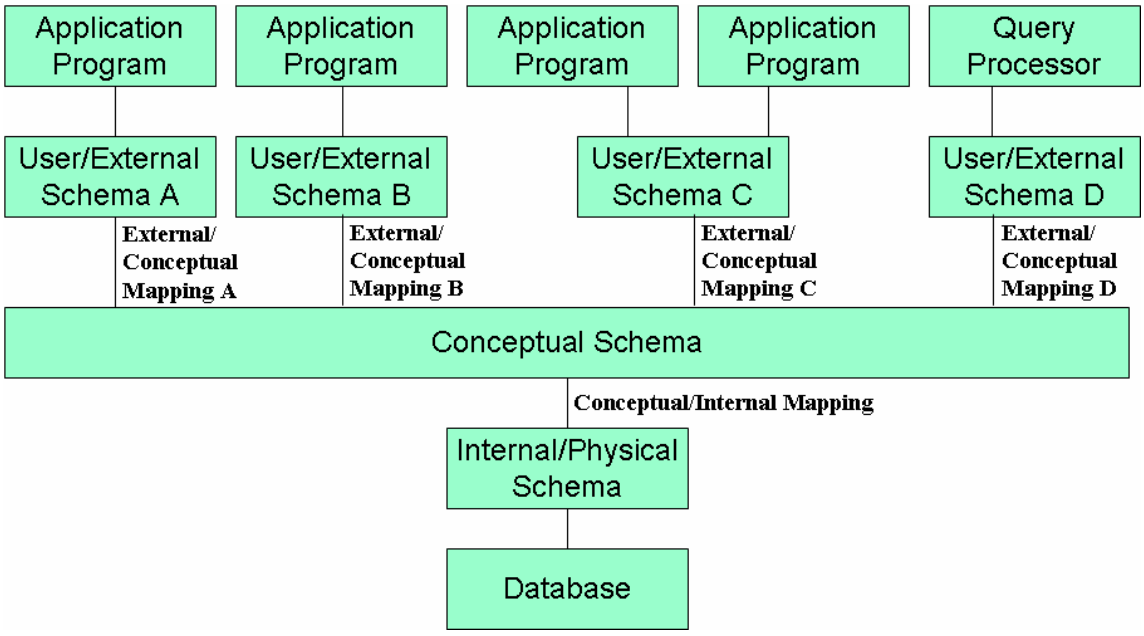
decision-support database may be needed to monitor recruitment and retention patterns among a student population.

3. MASS-DEPLOYMENT DATABASES

Such 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. 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 to build necessary update strategies between each type.

The ***data management layer*** is normally made up of the facilities provided by some DBMS. A DBMS can be considered as a buffer between application programs, end-users and a database designed to fulfill features of data independence. In 1975 the American National Standards Institute Standards Planning and Requirements Committee (ANSI-SPARC) proposed a three-level architecture for this buffer. This architecture identified three levels of abstraction. These levels are sometimes referred to as schemas or views:



- *The external or user level.* This level describes the users' or application program's view of the database. Several programs or users may share the same view.

- *The conceptual level*. This level describes the organizations' view of all the data in the database, the relationships between the data and the constraints applicable to the database. This level describes a logical view of the database – a view lacking implementation detail.

- *The internal or physical level*. This level describes the way in which data is stored and the way in which data may be accessed. This level describes a physical view of the database.

Each level is described in terms of a schema – a map of the database. The three-level architecture is used to implement data independence through two levels of mapping: that between the external schema and the conceptual schema, and that between the conceptual schema and the physical schema. Data independence comes in two forms:

- *Logical data independence*. This refers to the immunity of external schemas to changes in the conceptual schema. For instance, we may add a new data item to the conceptual schema without impacting on the external level.

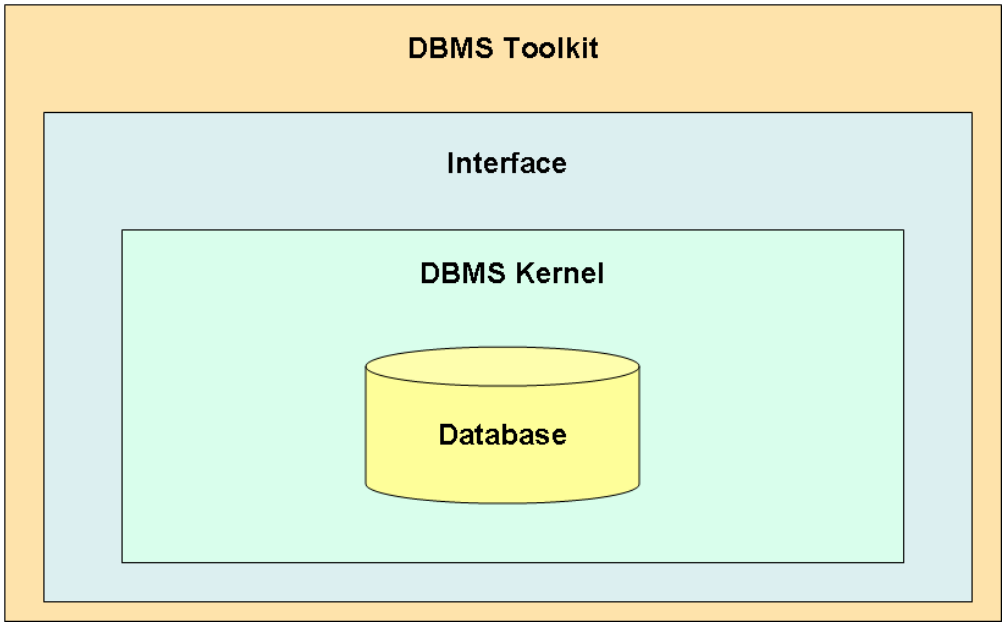
- *Physical data independence*. This refers to the immunity of the conceptual schema to changes made to the physical schema. For instance, we may change the storage structure of data within the database without impacting on the conceptual schema.

Any DBMS is composed of kernel, toolkit and exposed interface:

- *Kernel*. By DBMS kernel we mean the central engine which operates core data management functions such as most of those defined below.

- *Toolkit*. By DBMS toolkit we mean the vast range of tools which are now either packaged as part of a DBMS or are provided by third-party vendors. For instance, spreadsheets, fourth generation languages, performance monitoring facilities etc.

- *Interface*. Interposing between the kernel and toolkit we must have a defined interface. The interface is a standard language which connects a tool such as a fourth generation language with kernel functions.



The ***DBMS interface*** comprises a database sublanguage. A database sublanguage is a programming language designed specifically for initiating DBMS functions. Such a database sublanguage contains one or more of the following parts:

- *DATA DEFINITION LANGUAGE (DDL).* The data definition language is used to create structures for data, delete structures for data and amend existing data structures. The DDL updates the metadata held in the data dictionary.

- *DATA MANIPULATION LANGUAGE (DML).* The data manipulation language is used to specify commands which implement the CRUD activities against a database. The DML is the primary mechanism used for specifying transactions against a database.

- *DATA INTEGRITY LANGUAGE (DIL).* The data integrity language is used to specify integrity constraints. Constraints specified in a DIL are written to the data dictionary.

- *DATA CONTROL LANGUAGE (DCL).* The data control language is that part of the database sublanguage designed for use by the database administrator. It is particularly used to define authorized users of a database and the privileges associated with such users.

The dominant example of such a database sublanguage currently is the structured query language or SQL. Such a sublanguage is frequently used in association with some application-building tools to implement an ICT system.

The ***DBMS kernel*** is the glue between the DBMS and some native operating system. The DBMS needs to interact with the operating system, particularly to implement and access the database and system catalog which are usually stored on disk devices. Three elements of the operating system are critical to this interaction:

- *File manager*. The file manager of the operating system translates between the data structures manipulated by the DBMS and the files on disk. To perform this process it establishes a record of the physical structures on disk.

- *Access mechanisms*. The file manager does not manage the physical input and output of data directly. Instead it interacts with appropriate access mechanisms established for different physical structures.

- *System buffers*. The reading and writing of data are normally conducted in terms of the system buffers in the operating system. These are temporary structures for storing data and for manipulating data.

Modern DBMS are highly complex pieces of software. The actual architecture of a DBMS varies among products. However, it is possible to abstract from the variety of implementations a number of generic software components available in DBMS:

- *Database and system catalog*. The database and system catalog store the data and meta-data of an application.

- *Operating system*. Access to disk devices is primarily controlled by the host operating system which schedules disk input and output.

- *Database manager*. A higher-level data manager module in the DBMS controls access to the DBMS information stored on disk. All access to database information is performed under the control of the database manager. This database manager may use lower-level data management functions provided by the host operating system in its file management system or may use its own routines. Such functions are involved with transferring data to and from main memory and disk

- *DDL compiler*. The DDL compiler takes statements expressed in the Data Definition Language and updates the system catalog. All DBMS modules that need information about database objects must access the catalog.

- *Run-time processor*. The run-time processor handles retrieval or update operations expressed against the database. Access to disk goes through the data manager.

- *Query processor*. The query processor handles interactive queries expressed in a data manipulation language (DML) such as SQL. It parses and analyses a query before generating calls to the run-time processor.

- *Precompiler, DML compiler, host-language compiler*. The precompiler strips DML commands from an application program written in some host language. These commands are then passed to the DML compiler which builds object code for database access. The remaining parts of the application program are sent to a conventional compiler, and the two parts are linked to form a transaction which can be executed against the database.

The ***database management module*** of a DBMS enables the following functions:

- *Authorization control*. The database manager must check the authorization of all attempts to access the database.
- *Processing commands*. The database manager must execute the commands of the database sublanguage.
- *Integrity checking*. The database manager must check that every operation that attempts to make a change to the database does not violate any defined integrity constraints.
- *Query optimization*. This function determines the optimal strategy for executing any queries expressed against a database.
- *Managing transactions*. The database manager must execute any transactions against a database.
- *Scheduling transactions*. This function ensures the operation of concurrent access of transactions against a database.
- *Managing recovery*. This function is responsible for the commitment and aborting of transactions expressed against a database.
- *Managing buffers*. This function is responsible for transferring data to and between the main memory and the secondary storage.

As an overview, the key functions that any DBMS must support are:

1. CRUD FUNCTIONS

A DBMS must enable users to create data structures, to retrieve data from these data structures, to update data within these data structures and to delete data from these data structures. These functions are known collectively as CRUD activities – Create, Read, Update and Delete.

2. DATA DICTIONARY

A DBMS must support a repository of meta-data – data about data. This repository is known as a data dictionary or system catalog. Typically this data dictionary stores data about the structure of data, relationships between data-items, integrity constraints expressed on data, the names and authorization privileges associated with users. The data dictionary effectively implements the three-level architecture of a DBMS.

3. TRANSACTION MANAGEMENT

A DBMS must offer support for the concept of a transaction and must manage the situation of multiple transactions impacting against a database. A transaction is a series of actions which access or cause changes to be made to the data in a database.

4. CONCURRENCY CONTROL

A DBMS must enable many users to share data in a database – to access data concurrently. The DBMS must ensure that if two transactions are accessing the same data then they do not leave the database in an inconsistent state.

5. RECOVERY

The DBMS must ensure that the database is able to recover from hardware or software failure which causes the database to be damaged in some way.

6. AUTHORISATION

The DBMS must provide facilities for the enforcement of security. Generally the DBMS must support the concept of an authorized user of a database and should be able to associate with each user privileges associated with access to data within the database and/or facilities of the DBMS.

7. DATA COMMUNICATION

A DBMS must be able to integrate with communications software running in the context of an ICT system. This is particularly important to enable the connection of toolkit software with the kernel of a DBMS.

8. DATA INTEGRITY

Data integrity is that property of a database which ensures that it remains an accurate reflection of its universe of discourse. To enable this DBMS must provide support for the construct of an integrity constraint. The DBMS must be able to enforce such constraints in the context of CRUD activities.

9. ADMINISTRATION UTILITIES

The DBMS should ensure that there are sufficient facilities available for the administration of a database. Such facilities include:

- Facilities for importing data into a database from other data sources
- Facilities for exporting data from a database into other data sources
- Facilities for monitoring the usage and operation of a database
- Facilities for monitoring the performance of a database and for enhancing this performance

1.2. Relational Databases

The ***relational model*** is the basis for any ***relational database management system*** (***RDBMS***). A relational model has three core components: a collection of objects or relations, operators that act on the objects or relations, and data integrity methods. In other words, it has a place to store the data, a way to create and retrieve the data, and a way to make sure that the data is logically consistent.

The relational model was first proposed by Dr. E. F. Codd in 1969, in a paper called "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks". At that time, databases were primarily either of the hierarchical or network type.

We already presented the hierarchical and network models. Hierarchical and network-based databases are still used for extremely high-volume transaction-processing systems. IBM claims that 95% of the Fortune 1000 companies in the world still use IMS, a hierarchical database management system that is also web-enabled.

A ***relational database*** represents a collection of tables that stores data without any assumptions as to how the data is related within the tables or between the tables. A relational database uses relations, or two-dimensional tables, to store the information needed to support a business.

A ***table*** represents the basic construct of a relational database that contains rows and columns of related data. A table in a relational database, alternatively known as a ***relation***, is a two-dimensional structure used to hold related information. A database consists of one or more related tables.

NOTE: Don't confuse a relation with relationships. A relation is essentially a table, and a relationship is a way to correlate, join, or associate the two tables.

A ***row*** in a table is a collection or instance of one thing, such as one employee or one line item on an invoice. A ***column*** contains all the information of a single type, and the piece of data at the intersection of a row and a column, a ***field***, is the smallest piece of information that can be retrieved with the database's query language (such as SQL). For example, a table with information about employees might have a column called LAST_NAME that contains all of the employees' last names. Data is retrieved from a table by filtering on both the row and the column.

NOTE: SQL, which stands for Structured Query Language, supports the database components in virtually every modern relational database system. SQL has been refined and improved by the American National Standards Institute (ANSI) for more than 20 years. As of Oracle9i, Oracle's SQL engine conforms to the ANSI SQL: 1999 (also known as SQL3) standard, as well as its own proprietary SQL syntax that existed in previous versions of Oracle. Until Oracle9i, only SQL: 1992 (SQL2) syntax was fully supported.

A relational database can enforce the rule that fields in a column may or may not be empty. Let us consider the example of the EMP table that holds the basic employee information, presented in the following figure:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Notice that some fields in the Commission (COMM) and Manager (MGR) columns do not contain a value; they are blank. In this case, it makes sense for an employee who is not in the Sales department to have a blank Commission field. It also makes sense for the president of the company to have a blank Manager field, since that employee doesn't report to anyone.

On the other hand, none of the fields in the Employee Number (EMPNO) column are blank. The company always wants to assign an employee number to an employee, and that number must be different for each employee. One of the features of a relational database is that it can ensure that a value is entered into this column and that it is unique. The EMPNO column, in this case, is the ***primary key*** of the table.

The ***primary key*** represents a column (or columns) in a table that makes the row in the table distinguishable from every other row in the same table.

As you might suspect, the DEPTNO column contains the department number for the employee. But how do you know what department name is associated with what number? The user created the DEPT table to hold the descriptions for the department codes in the EMP table.

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

The DEPTNO column in the EMP table contains the same values as the DEPTNO column in the DEPT table. In this case, the DEPTNO column in the EMP table is considered a ***foreign key*** to the same column in the DEPT table. A foreign key represents a column (or columns) in a table that draws its values from a primary or unique key column in another table. A foreign key assists in ensuring the data integrity of a table. With this association, we can enforce the restriction that a DEPTNO value cannot be entered in the EMP table unless it already exists in the DEPT table. A foreign key enforces the concept of ***referential integrity*** in a relational

database. A referential integrity is a method employed by a relational database system that enforces one- to-many relationships between tables. The concept of referential integrity not only prevents an invalid department number from being inserted into the EMP table, but it also prevents a row in the DEPT table from being deleted if there are employees still assigned to that department.

Before the user created the actual tables in the database, he went through a design process known as ***data modeling*** (a process of defining the entities, attributes, and relationships between the entities in preparation for creating the physical database.) In this process, the developer conceptualizes and documents all the tables for the database. One of the common methods for modeling a database is called ERA, which stands for entities, relationships, and attributes. The database designer uses an application that can maintain entities, their attributes, and their relationships. In general, an entity corresponds to a table in the database, and the attributes of the entity correspond to columns of the table.

NOTE: Various data modeling tools are available for database design. Examples include Microsoft Visio and more robust tools such as Computer Associate's ERwin and Embarcadero's ER/Studio.

The data-modeling process involves defining the entities, defining the relationships between those entities, and then defining the attributes for each of the entities. Once a cycle is complete, it is repeated as many times as necessary to ensure that the designer is capturing what is important enough to go into the database.

First, the designer identifies all of the entities within the scope of the database application. The entities are the persons, places, or things that are important to the organization and need to be tracked in the database. Entities will most likely translate neatly to database tables. For example, for the first version of the widget company database, he identifies four entities: employees, departments, salary grades, and bonuses. These will become the EMP, DEPT, SALGRADE, and BONUS tables.

Once the entities are defined, the designer can proceed with defining how each of the entities is related. Often, the designer pairs each entity with every other entity and ask, "Is there a relationship between these two entities?" Some relationships are obvious; some are not.

In the widget company database, there is most likely a relationship between EMP and DEPT, but depending on the business rules, it is unlikely that the DEPT and SALGRADE entities are related. If the business rules were to restrict certain salary grades to certain departments, there would most likely be a new entity that defines the relationship between salary grades and departments. This entity would be known as an ***associative*** or ***intersection table***, and would contain the valid combinations of salary grades and departments.

An associative table represents a database table that stores the valid combinations of rows from two other tables and usually enforces a business rule. An associative table resolves a many-to-many relationship.

In general, there are three types of relationships in a relational database:

The most common type of relationship is ***one-to-many***. A one-to-many relationship represents a relationship type between tables where one row in a given table is related to many other rows in a child table. The reverse condition, however, is not true. A given row in a child table is related to only one row in the parent table. This means that for each occurrence in a given entity, the parent entity, there may be one or more occurrences in a second entity, the child entity, to which it is related. For example, in the widget company database, the DEPT entity is a parent entity, and for each department, there could be one or more employees associated with that department. The relationship between DEPT and EMP is one-to-many.

In a ***one-to-one*** relationship, a row in a table is related to only one or none of the rows in a second table. These relationships are not as common as one-to-many relationships, because if one entity has an occurrence for a corresponding row in another entity, in most cases, the attributes from both entities should be in a single entity. A one-to-one relationship represents a relationship type between tables where one row in a given table is related to only one or zero rows in a second table. This relationship type is often used for subtyping. For example, an EMPLOYEE table may hold the information common to all employees, while the FULLTIME, PARTTIME, and CONTRACTOR tables hold information unique to full time employees, part time employees and contractors respectively. These entities would be considered subtypes of an EMPLOYEE and maintain a one-to-one relationship with the EMPLOYEE table.

In a ***many-to-many*** relationship, one row of a table may be related to many rows of another table, and vice versa. Usually, when this relationship is implemented in the database, a third entity is defined as an intersection table to contain the associations between the two entities in the relationship. For example, in a database used for school class enrollment, the STUDENT table has a many-to-many relationship with the CLASS table—one student may take one or more classes, and a given class may have one or more students. The intersection table STUDENT_CLASS would contain the combinations of STUDENT and CLASS to track which students are in which class. A many-to-many relationship represents a relationship type between tables in a relational database where one row of a given table may be related to many rows of another table, and vice versa. Many-to-many relationships are often resolved with an intermediate *associative table.*

Once the designer has defined the entity relationships, the next step is to assign the attributes to each entity. This is physically implemented using columns, as shown here for the SALGRADE table as derived from the salary grade entity.

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999
6	10000	12500

After the entities, relationships, and attributes have been defined, the designer may iterate the data modeling many more times. When reviewing relationships, new entities may be discovered. For example, when discussing the widget inventory table and its relationship to a customer order, the need for a shipping restrictions table may arise.

Once the design process is complete, the physical database tables may be created. This is where the DBA usually steps in, although the DBA probably has attended some of the design meetings already! It's important for the DBA to be involved at some level in the design process to make sure that any concerns about processor speed, disk space, network traffic, and administration effort can be addressed promptly when it comes time to create the database.

Logical database design sessions should not involve physical implementation issues, but once the design has gone through an iteration or two, it's the DBA's job to bring the designers "down to earth." As a result, the design may need to be revisited to balance the ideal database implementation versus realities of budgets and schedules.

1.3. Database Administration

Database administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software hire specialized IT personnel called Database Administrators or DBAs. This course introduces you to the universe of database administration; it helps you to become qualified DBA practitioners.

There are three types of DBAs:

1. ***Systems DBAs*** (sometimes also referred to as Physical DBAs, Operations DBAs or Production Support DBAs): usually focus on the physical aspects of database administration such as DBMS installation, configuration, patching, upgrades, backups, restores, refreshes, performance optimization, maintenance and disaster recovery.

2. ***Development DBAs***: usually focus on the logical and development aspects of database administration such as data model design and maintenance, DDL (data definition language) generation, SQL writing and tuning, coding stored procedures, collaborating with developers to help choose the most appropriate DBMS feature/functionality and other pre-production activities.

3. ***Application DBAs***: are usually found in organizations that have purchased 3rd party application software such as ERP (enterprise resource planning) and CRM (customer relationship management) systems. Examples of such application software include Oracle Applications, Siebel and PeopleSoft (both now part of Oracle Corp.) and SAP. Application DBAs straddle the fence between the DBMS and the application software and are responsible for ensuring that the application is fully optimized for the database and vice versa. They usually manage all the application components that interact with the database and carry out activities such as application installation and patching, application upgrades, database cloning, building and running data cleanup routines, data load process management, etc.

While individuals usually specialize in one type of database administration, in smaller organizations, it is not uncommon to find a single individual or group performing more than one type of database administration. Database administration work is complex, repetitive, time-consuming and requires significant training. Since databases hold valuable and mission-critical data, companies usually look for candidates with good experience. Database administration often requires DBAs to put in work during off-hours (for example, for planned after hours downtime, in the event of a database-related outage or if performance has been severely degraded). DBAs are commonly well compensated for the long hours.

Often, the DBMS software comes with certain tools to help DBAs manage the DBMS. Such tools are called native tools. For example, Microsoft SQL Server comes with SQL Server Enterprise Manager and Oracle has tools such as SQL*Plus and Oracle Enterprise Manager/Grid Control. In addition, 3rd parties such as BMC, Quest Software, Embarcadero and SQL Maestro Group offer GUI tools to monitor the DBMS and help DBAs carry out certain functions inside the database more easily.

Another kind of database software exists to manage the provisioning of new databases and the management of existing databases and their related resources. The process of creating a new database can consist of hundreds or thousands of unique steps from satisfying prerequisites to configuring backups where each step must be successful before the next can start. A human cannot be expected to complete this procedure in the same exact way time after time - exactly the goal when multiple databases exist. As the number of DBAs grows, without automation the number of unique configurations frequently grows to be costly/difficult to support. All of these complicated procedures can be modeled by the best DBAs into database automation software and executed by the standard DBAs.

Recently, automation has begun to impact this area significantly. Newer technologies such as HP/Opsware's SAS (Server Automation System) and Stratavia's Data Palette suite have begun to increase the automation of servers and databases respectively causing the reduction of database related tasks. However at best this only reduces the amount of mundane, repetitive activities and does not eliminate the need for DBAs. The intention of DBA automation is to enable DBAs to focus on more proactive activities around database architecture and deployment.

There are three common types of users of database systems – end-users, database administrators (DBAs) and systems developers. Each type of user demands a different type of interface to the database system:

- *End-users*. Most users in organizations are unlikely to know they are using a database system. This is because they are using a database system indirectly through some ICT system. Some sophisticated end-users may be able to access database systems directly through employing end-user tools such as visual query interfaces.

- *Database administrators/data administrators*. DBAs are concerned with creating and maintaining databases for various applications. Hence, most DBMS provide a range of tools for DBAs, particularly in the area of data control.

- *System developers*. Developers of ICT systems need to integrate database systems with the wider functions of the ICT system. Various tools such as application programming interfaces are available for this purpose.

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 DBA works primarily at the physical level. The place where the data administrator and the DBA meet is at the logical level. Both the data administrator and DBA 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 a database is being used by many people and the volume of data is significant, the need for a person or persons who specialize in administration functions becomes apparent.

In many organizations, particularly small organizations, the DBA is expected to undertake many data administration tasks. However, in general, the DBA can be said to have the following core responsibilities: administration of the database, administration of the DBMS and administration of the database environment.

1. ADMINISTRATION OF THE DATABASE

The DBA would normally be expected to engage in the following key activities in relation to administering a database or a series of databases:

- *Physical design* – whereas the data administrator is concerned with the conceptual and logical design of database systems, the DBA is concerned with the physical design and implementation of databases.

- *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.

- *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.

- *Data archiving* – establishing a strategy for archiving of ‘dead’ data.

- *Data backup and recovery* – establishing a procedure for backing-up data and recovering data in the event of hardware or software failure.

2. ADMINISTRATION OF THE DBMS

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

- *Installation* – taking key responsibility for installing DBMS or DBMS components.
- *Configuration control* – enforcing policies and procedures for managing updates and changes to the software of the database system.
- *Monitoring DBMS usage and tuning DBMS* – monitoring live running of database systems and tailoring elements of the DBMS structure to ensure the effective performance of such systems.

3. 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:

- *Data control* – establishing user groups, assigning passwords, granting access to DBMS facilities, granting access to databases.
- *Impact assessment* – assessing the impact of any changes in the use of data held within database systems.
- *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.
- *Training* – holding responsibility for the education and training of users in the principles and policies of database use

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 preeminent 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 DBA.

We shall examine some of the fundamental aspects of data control in relational database systems. First, we shall examine the concept of a relational view in SQL. Second, we shall discuss how DBAs manipulate the view concept in order to allocate access to data. Third, we shall examine the DBA's ability to allocate access to DBMS functions such as data definition.

1. 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.

EXAMPLE For instance, suppose we have a base table of student data, a sample of which is given below:

Students

Student no	Student name	Sex	Date of Birth	Student Addr	Student Tel	Course Code
1234	Paul S	M	01/03/1980	14, Hillside Tce	432568	CSD
1235	Patel J	M	01/05/1981	28, Park Place	235634	CSD
2367	Singh B	F	01/01/1974	10a, Pleasant Flats	345623	BSD
3245	Conran J	M	01/08/1980	Rm 123, University Towers	NULL	CSD
2333	Jones L	F	01/02/1981	9 Redhill St	222333	BSD

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

a. SIMPLIFICATION

Suppose the students table above is defined for an entire university. Each department or school within the university however only wishes to see data relevant to its own courses. A number of views might then be created to simplify the data that each department or school sees.

EXAMPLE For instance, we might have two views established on the table above, one for the students of the school of computing and one for the students of the business school.

Computing Students

Student No	Student name	Sex	Student Addr	Student Tel
1234	Paul S	M	14, Hillside Tce	432568
1235	Patel J	M	28, Park Place	235634
3245	Conran J	M	Rm 123, University Towers	NULL

Business Students

Student No	Student name	Sex	Student Addr	Student Tel
2367	Singh B	F	10a, Pleasant Flats	345623
2333	Jones L	F	9 Redhill St	222333

b. FUNCTIONALITY

There is an implicit edict in the relational model: store as little as possible. One consequence of this is that any data that can be derived from other data in the database is not normally stored.

EXAMPLE If, for instance, there is a requirement to regularly display the ages of students, we would not store both a student's age and his or her birthdates. A student's age can be derived from a person's birthdates. The view mechanism is the conventional means for performing such computations and displaying them to the user. For example, in the view below we display the sex and age profile of students. Age is a derived attribute in this view.

Students

Student no	Student name	Sex	Age

1234	Paul S	M	19
1235	Patel J	M	18
2367	Singh B	F	25
3245	Conran J	M	18
2333	Jones L	F	19

c. SECURITY

In a university we might want to restrict each department or school only to 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.

Retrieving information through views is a straightforward process. It involves merging the original query on the view with the view definition itself. File maintenance operations such as insertion, deletion and update are however more complex to accomplish via views, and hence deserve more consideration. The main point is that not all views are updatable.

EXAMPLE Consider for instance, two views declared on the students table, S1 and S2. S1 produces a view of all the columns in the base table but only for computing students; S2 is a vertical fragment of the base table:

S1

Student no	**Student name**	**Sex**	**Date of Birth**	**Student Addr**	**Student Tel**	**Course Code**
1234	Paul S	M	01/03/1980	14, Hillside Tce	432568	CSD
1235	Patel J	M	01/05/1981	28, Park Place	235634	CSD

3245	Conran J	M	01/08/1980	Rm 123, University Towers	NULL	CSD

S2

Student name	**Sex**	**Course Code**
Paul S	M	CSD
Patel J	M	CSD
Singh B	F	BSD
Conran J	M	CSD
Jones L	F	BSD

In the case of **S1**, for instance, we can insert a new row into the view, delete an existing record from a view or update an existing field in the view. In the case of **S2**, problems arise. For example, suppose we wished to insert a record for a new student, Balshi J, M, BSD. We would have to insert the record *null*, *Balshi J*, *M*, *null*, *null*, *BSD* into the underlying students table. This attempt will fail since studentNo (the table's primary key) must not be null.

On closer examination, the main difference between **S1** and **S2** lies in the fact that **S1** contains the primary key column of the base table and **S2** does not. This means that updates to **S1** unambiguously update rows in a view. Updates to **S2** are frequently ambiguous. **S1** is hence said to be an updatable view whereas S2 is a non-updatable view.

As stated previously, 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. A view in SQL is a packaged select statement. The syntax is as follows:

CREATE VIEW <view name> AS <select statement>

EXAMPLE The following two examples create the **S1** and **S2** views:

*CREATE VIEW S1 AS SELECT * FROM Students WHERE courseCode = 'CSD'*

CREATE VIEW S2 AS SELECT studentName, sex, courseCode FROM Students

When a view has been created, it can be queried in the same manner as a base table. For instance:

*SELECT * FROM S1 WHERE sex = 'F'*

C. Date has made the distinction between three types of views (Date, 2000):

- Views which are updatable in theory and in practice
- Views which are updatable in theory, but not yet in practice
- Views which do not appear to be updatable in theory and therefore cannot be updatable in practice

What then defines the first type of view? As far as most SQL-based systems are concerned, the following conditions have to be satisfied:

- The *select* clause in a view cannot use a distinct qualifier or any function definition
- The *from* clause must specify just one table. No joins are allowed
- The *where* clause cannot contain a correlated subquery
- The *group by* and *order* by clauses are not allowed
- No *union* operator is permitted

In any database system, the DBA needs to be able to do three main things:

- Prevent would-be users from logging-on to the database
- Allocate access to specific parts of the database to specific users
- Allocate access to specific operations to specific users

We shall examine SQL's capability for handling the second of these needs. Because the first and third functions are non-standard across DBMS we shall resort to examining the facilities available under the ORACLE RDBMS.

1. GRANT AND REVOKE

SQL provides two commands for controlling access to data. One command gives access privileges to users. The other command takes such privileges away.

To give a user access to a view or a table we use the grant command:

GRANT [ALL : SELECT : INSERT : UPDATE : DELETE]

ON [<table name> : <view name>]

TO <user name>

EXAMPLE For instance:

GRANT INSERT ON Modules TO pbd

GRANT SELECT ON Lecturers TO pbd

To take existing privileges away from a user we use the revoke command.

REVOKE [ALL : SELECT : INSERT : UPDATE : DELETE]

ON [<table name> : <view name>]

FROM <user name>

EXAMPLE For instance:

REVOKE INSERT ON Modules FROM pbd

REVOKE SELECT ON Lecturers FROM pbd

2. GRANTING ACCESS VIA VIEWS

One of the ways of enforcing control in relational database systems is via the view concept.

EXAMPLE Let us suppose, for instance, that we wish to give R. Evans the ability to look at and update the records of Lecturers in his department – Computer Studies. We do not however wish to give R. Evans the ability to delete Lecturer records or insert new Lecturer records. To enact these organization rules we create a view:

CREATE VIEW Evans AS

*SELECT * FROM Lecturers*

WHERE deptName =

(SELECT deptName FROM Lecturers WHERE staffName = 'Evans R')

Note that the employee record for R. Evans would be included in this view. If we now provide select and update access for R. Evans as below:

GRANT SELECT, UPDATE ON research TO REvans

This particular person will be able to change his own salary! There are a number of ways we can prevent this from happening by modifying the view definition. One possible solution is given below:

CREATE VIEW Evans AS

*SELECT * FROM Lecturers*

WHERE deptName =

(SELECT deptName FROM Lecturers WHERE staffName = 'Evans R')

AND staffName < > 'Evans R'

3. DBA PRIVILEGES IN ORACLE

In the above discussion we have considered how SQL permits the DBA to specify access to parts of the database and how it permits the DBA to associate these allocations with particular users. However, at present SQL does not contain a definition of how users are declared to the database system in the first place, and also how such users are given access to various DBMS facilities. Assigning users and giving such users access privileges are two DBA functions which vary from product to product. In this course ORACLE will be extensively used for examples. To give the reader a flavor of the type of mechanisms available (more will be discussed in latter course lesson), we discuss here those available under the ORACLE RDBMS.

4. CONNECT, RESOURCE AND DBA

Most RDBMS differ in the levels of facilities or system privileges offered and what they are called. In this section we illustrate what is available under the ORACLE DBMS.

When a company receives a new ORACLE system it comes automatically installed with two 'super' users. The first thing the DBA does is change the default passwords for these user names. The second thing he or she does is start enrolling users into the system.

To declare an ORACLE user the DBA must normally supply three pieces of information: a distinct user name, a password and the level of privilege granted to the user.

For now you should remember that there are three classes of ORACLE user: *connect*, *resource* and *dba*. A connect user is able to look at other users' data only if allowed by other users, perform data manipulation tasks specified by the DBA, and create views. Resource privilege allows the user to create database tables and indexes and grant other users access to these tables and indexes. Dba privilege is normally given to a chosen few. The superusers discussed above have dba privileges. Such privileges permit access to any user's data, and allow the granting and revoking of access privileges to any user in the database.

When a new user is enrolled into the system the database administrator grants the user one or more of these privileges using the command below:

GRANT {CONNECT : RESOURCE : DBA}

TO <username>

[IDENTIFIED BY <password>]

EXAMPLE For instance:

GRANT CONNECT, RESOURCE TO pbd IDENTIFIED BY crs

In a similar manner to granting database access, DBMS privileges can be revoked as below:

REVOKE {CONNECT : RESOURCE : DBA} FROM <username>

EXAMPLE For instance:

REVOKE CONNECT, RESOURCE FROM pbd

In the next chapters of the course we will discussed in more details the activities specific to a DBA. For now consider this:

- The database administrator is a low-level, technical function
- In terms of development, the database administrator will be involved in the physical design of database systems. On the data management side, the database administrator will be particularly concerned with the issue of data control
- Data control is the activity devoted to controlling access to data and DBMS functions
- In relational database systems, the data aspect of data control revolves around the concept of a view. A view is a virtual table. Views can be used to create 'windows' into a database which can be allocated to specific users of the database
- The DBMS-side of data control varies from product to product. Most RDBMS however have facilities for assigning user names and passwords to users, and defining the overall privileges available to users
- ISO SQL has a limited range of DBA functions, particularly focused on granting access to data and revoking access to data. Most DBMS also have a range of nonstandard functions for declaring users and passwords, fine-tuning database sizing, monitoring the performance of a database and fragmenting databases

References

[1] Paul Beynon-Davies, "*Database Systems. Third Edition.*", Published by Palgrave Macmillan, 2004.

[2] Bob Bryla, "*Oracle 9i DBA Jump Start*", Sybex, 2003.

[3] Gavin Powell, "*Beginning Database Design*", Wiley Publishing, 2006.

[4] Michael J. Hernandez, "Proiectarea Bazelor de Date", Teora, 2003.