Databases

Susanne Bleisch

June 8, 2016

Contents

1 Introduction to Database Systems

data management and especially the management of geodata is not bound to a specific technology. It would be possible to use analogous map archives or file based record systems. However, the term Geoinformation System implies some demands that exceed the storage and retrieval of data. These additional needs can be satisfied sensibly with database systems only.

This lesson is focused on database concepts and architectures. After an introduction and the definition of some of the most important terms in the unit Definition of Terms (Page 7) we will devote ourseleves to the specific characteristics of the database approach (unit Characteristics of the Database Approach (Page 12)). The closer examination of various applications of databases in the unit Example Applications (Page 20) will allow you to become acquainted with the use of databases in different fields and contexts and to extend the knowledge about the characteristics of databases. A comparison of the database approach with file based solutions is made in the unit Advantages and Disadvantages (Page 29).

Learning Objectives

- Be used to the terminology of data, information and data management and be able to explain the most important terms.
- Know and understand the characteristics of database systems and be able to transfer these to applications in your daily life.

1.1 Definition of Terms

From Data to usable Information...

Before we start with the use and architecture of data management solutions we discuss the terminology of this field to get to know the basic terms like information, data, database systems. A lot of these basic terms are used daily but most often not in the right context.

1.1.1 Data versus Information

Data (especially computer data): The presentation of facts, information or concepts which are created in a computer readable form or are translated into such a form.

Information: Information is a useable answer to a concrete question. [**?**]

Something is information if a specific question is answered and that answer increases the understanding of the questioner and enables them to come closer to a specific objective. (Translation from [**?**])

Information has the following aspects:

- *structured*[1] and *syntactic*[2]
- *semantic* (as regards content)
- *pragmatic* (relevant to applications)

Relationship between Data and Information

The terms data and information are often used interchangeably and in the wrong context. Therefore a list of distinguishing features is presented below

- Semantic aspects of data are often coded. These codes need to be defined and interpreted after conventions previously agreed upon (e.g. Grades from 1 to 6 with the convention 6 = very good).
- Generally, information needs to be reconstructed or derived from data (e.g. the average rainfall of the month July over the last 10 years).
- Normally, data do not contain aspects relevant to applications (e.g. it is not possible to derive information for applications like tax, developement, flood risk, etc. from the coordinates of a parcel of land).

[1] IIn the information theory three dimensions of information are distinguished: the syntactic, the semantic and the pragmatic. Let us take the example of a traffic light. In the syntactic dimension we differentiate the three colours red, yellow and green. But the traffic light makes more sense in the semantic dimension. In this dimension the colours are linked to meanings. Red means stop, green means go. However, only in the pragmatic dimension does the traffic light become useable for the traffic. Pragmatically, red means that the driver of a car must stop.

[2] Syntax can in linguistics be described as the study of the rules, or "patterned relations" that govern the way the words in a sentence come together.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 1: A letter in chinese language

Trying to read a letter in a foreign language, we are able to recognise the structure and the syntactic aspects of it like the paragraphs, sentences, words, etc. but we cannot make out the meaning of the writing.

However, writing in our own language, cannot be called information in every case. At least we might be able to understand the content (*semantic*) but if this content is irrelevant or not interesting to us then the important aspect of the usefulness is missing.

1.1.2 The Components of an Information System

Conceptually[3] an information systems has a layered structure.

Move your mouse over the terms of the following interaction and get to know what parts make up an information system.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 2: The Components of an Information System

The data management components of an information system are:

Data and Database: An amount of data which are viewed by the operator as somehow going together plus additional data which is used by the DBMS to work correctly are called a database.

Database Management System: A database management system (DBMS) is a software product for the persistent, consistent and application independent storage and management of data. But also for the flexible and easy use of big and concurrently used databases.

Database System: A database system (DBS) consists of a DBMS and one or more databases.

Database management systems and database systems are in the focus of this module.

Information System: An information system extends the database with a couple of software tools for querying, presenting, transforming and analysing the data.

According to the first part of this unit where the difference between data and information (Page 8) According to the first part of this unit where the difference between data and information were discussed, the tools of an information system enrich the data with *semantic* and *pragmatic* aspects.

[3] A concept is an abstract, universal idea, notion, or entity that serves to designate a category or class of entities, events, or relations. Concepts are abstract in that they omit the differences of the things in their extension, treating them as if they were identical. They are universal in that they apply equally to everything in their extension. Concepts are also the basic elements of propositions, much the same way a word is the basic semantic element of a sentence.

For sure, you have already heard the term geoinformation system and perhaps read the one or more definitions of it. The following paragraph defines the term geoinformation system and compares this definition with the one of an information system as discussed earlier in this unit. Geoinformation System:

Geoinformation System: "*A geoinformation system allows capturing, storing, analysing and presenting of all data that describe a part of the earth's surface and all on this part located technical and administrative equipment but also geoscientific, economic and ecologic features. (Translation)*" [**?**]

This definition contains the most important aspects of the definition of an information system but focuses on data with spatial referencing.

1.2 Characteristics of the Database Approach

The database approach has some very characteristic features which are discussed in detail in this unit.

In the unit Example Applications (Page 20) the use of databases in different fields is presented and according to these examples the most important features of the database approach revised.

A comparison between the file based approach and the database approach can be found in the unit Advantages and Disadvantages (Page 29).

1.2.1 Concurrent Use

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system.

Such concurrent use of data increases the economy of a system. Data capturing and data storage is not *redundant*[4], the system can be operated from a central control and the data can be updated more efficiently. Additionally, better use of the most often very expensive *(geo) data*[5] can be made.

When using data concurrently the problem of how the system should behave if changes are made simultanteously (e.g. two different users with different applications change the same data simultaneously) needs to be solved. Additionally, there is a serious security risk, for example, in the realms of data protection.

In technical jargon changes to a database are called transactions. This term is explained later in this lesson.

An example for concurrent use is the travel database of a large travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 3: Concurrent Use of the same Data

[4]Redundancy, in general terms, refers to the quality or state of being redundant, that is: exceeding what is necessary or normal, containing an excess. This can have a negative connotation, superfluous, but also positive, serving as a duplicate for preventing failure of an entire system.

[5]Geodata or data with a spatial relation are data about objects that through a position in space directly or indirectly can be referenced. The space is defined through a coordinate system which is in relation to the earth's surface.

1.2.2 Structured and Described Data

A fundamental feature of the database approach is that the database system does not only contain the data but also the complete definition and description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data").

Metadata[6] is used by the DBMS software but also by applications like GIS and by users of databases. As DBMS software is not written for one specific database application the *metadata* of a database is used to get information about the extent, the structure, etc. of it.

Structured Data: Data is called structured if it can be subdivided systematically and linked.

Following is a simple example how data can be described in a database.

Following is a simple example how data can be described in a database. Below there is a database table. Because of the structure of this table (first column = Forename, second column = Surname, third column = Postcode, forth column = City) it is known that a entry in the first column must be a forename (coded as string) and an entry in the third column must be a postcode (coded as number).

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 4: Example of an Database Table

[6]Metadata is literally "data about data", is information that describes another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: It is data about the data in the book referred to by the card. Other common contents of metadata include the source or author of the described dataset, how it should be accessed, and its limitations. Another important type of data about data is the links or relationship among data.

1.2.3 Separation of Data and Applications

As described in the feature structured data (Page 14), the structure of a database is described through *metadata* which is also stored in the database.

A software application does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system of a database (DBMS) via a standardised interface with the help of a standardised language like SQL. The access to the data and the metadata is entirely done by the DBMS.

In this way all the applications can be totally seperated from the data. Therefore database internal reorganisations or improvement of efficiency do not have any influence on the application software.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 5: Separation of Data and Applications

1.2.4 Data Integrity

Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorised access (confidentiality) and unauthorised changes.

Data reflect facts of the real world. Logically, it is demanded that this reflection is done correctly. A DBMS should support the task to bring only correct and consistent data into the database. Additionally, correct transactions (Page 17) ensure that the consistency is maintained during the operation of the system.

An example for inconsistency would be if contradictory statements were saved in the same database.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 6: Two Database Tables with Contradictory Datasets

1.2.5 Transactions

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state. In between the data are inevitable inconsistencies.

A transaction is atomic, which means it cannot be divided up any further. Within a transaction all or none of the actions need to be carried out. Doing only a part of the actions would lead to an inconsistent database state.

One example of a transaction is the transfer of an amount of money from one bank account to another. The debit of the money from one account and the credit of it to another account makes together a consistent transaction. This transaction is also atomic. The debit or credit alone would both lead to an inconsistent state. After finishing the transaction (debit and credit) the changes to both accounts become persistent and the one who gave the money has now less money on his account while the receiver has now a higher balance.

Try it using the buttons at the bottom to the left which allowing navigation through the steps of this example.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 7: Transaction of money from one account to another

1.2.6 Data Persistence

Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly be the user and must not be dependent on system features. Additionally data once stored in a database must not be lost.

Changes of a database which are done by a transaction (Page 17) are persistent. When a transaction is finished even a system crash cannot put the data in danger.

1.2.7 Data Views

Typically, a database has several users and each of them, depending on access rights and desire, needs an individual view of the data (content and form). Such a data view can consist of a subset of the stored data or from the stored data derived data (not explicitely stored).

A university manages the data about students. Beside matriculation number, name, address, etc. other information, for example in which course the student is registered, if he needs to do a resit, and so on is managed as well.

This extensive database is used by several people all with different needs and rights.

Please click on the four buttons below to see the different data views for different users of this database.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 8: Different Data Views

The database administrator has a view on the whole database while other users in this example have only a restricted view on the database. The administrator, for example, does not need information about names and matriculation numbers in case they want to create an anonymous statistic about the resits. In the student lists there should not be any sensitive data about the resits or similar. However, the lecturer of a class needs the detailed information about the students in this class including resits and so on.

1.3 Example Applications

Current information technology solutions distinguish themselves by having many distributed users which want to concurrently use constantly updated data. Therefore, database systems are used in different fields and there is rarely an information technology solution that gets by without one.

Following a couple of examples for the use of database systems are discussed.

For each example the application is breifly described first and then the for this application specific database features are commented on. The respective features with their descriptions can be found in the unit Characteristics of the Database Approach (Page 12) which is linked from the keywords.

1.3.1 Management of Bank Accounts

The management of bank accounts are a demanding task which has used database systems as aid for some time. Today, it is unthinkable to operate in the very complex world of finances without the help of database systems.

The most important characteristics of such database systems:

Transaction (Page 17)	The successfull and correct course of transactions is very important when managing bank accounts. It cannot be true that a credit is made to the wrong account or that a debit is made more than once.
Data Integrity (Page 16)	Data integrity is very important. It needs to be clearly defined what the requirements and rules of consistency are and how these can be followed.
Data Persistence (Page 18)	For an owner of a bank account it is reassuring to know that the persistence of the data is guaranteed. Data are not deleted arbitrarily or are lost mysteriously.

1.3.2 Timetable Informationsystem

The online timetable of the SBB (Swiss Federal Railways) is an example of a web based information system founded on a database system from the bounds of public transport. Its main task is to inform the users with current and correct information about the best connections and the train operation of the SBB at any time.

The most important characteristics of such a database system:

Concurrent Use (Page 13)	The database system of the SBB timetable can be used by different users and applications concurrently. While the information are used internally within the SBB, information about the timetables and other specific information can also be used from the public via the internet.
Data Integrity (Page 16)	The users of the SBB timetables need current and correct information at any time. This makes great demands on the data integrity. Therefore, timetable or platform changes or any other changes need to be updated constantly in the database system.

Such a database system can get more than one call per second in record times. Therefore, perfomance is also a very important characteristic which was not mentioned above.

Internet access to the SBB timetable: http://fahrplan.sbb.ch/ (fahrplan.sbb.ch/)

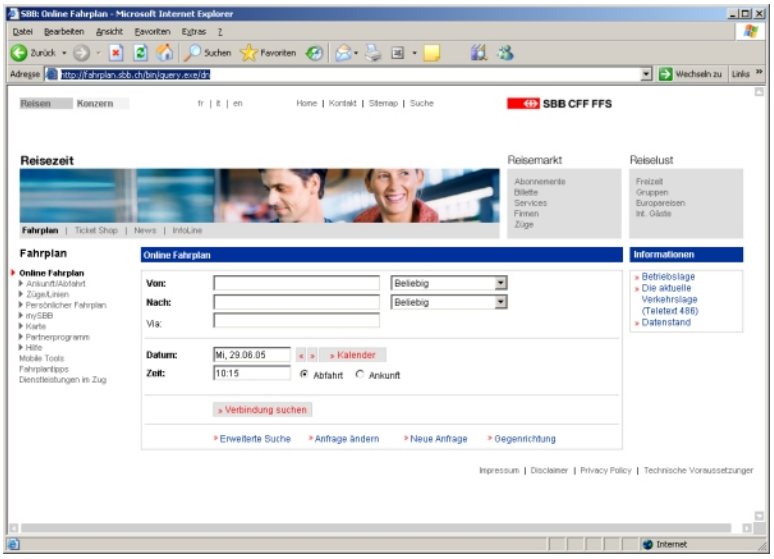

Figure 9: Screenshot SBB Timetable Informationsystem [**?**]

1.3.3 Library Catalogue

The library catalogue NEBIS [?] is the catalogue of a network of libraries and information desks in Switzerland. With the help of NEBIS it is possible to search for specific books or publications in libraries all over Switzerland.

The most important characteristics of such a database system:

Structured and Described Data (Page 14)	It is more than helpful to have a clearly defined structure when recording and updating several thousands of books, magazins and publications. Additionally, the description of the data allows to search selectively for specific objects. Further, a clear structure is also free of redundancy. This saves work as already a very small *redundancy* would lead to a multiple of work.
Data Views (Page 19)	Depending on how detailled and specialised a search should be done, a user needs more or less information from the database. Thus, for a coarse search title and author might already be sufficient. However, when looking for a specific edition of a book, some more information is needed. With this in view the user can choose between different data views. Additionally, there are views for the manager of the catalogue which are not accessible by the public user.

Different Data Views (click on the thumbnail for a bigger image)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 10: Result List of the Search [?]

Online access to the NEBIS library catalogue: http://www.nebis.ch/ (www.nebis.ch/)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 11: Standard View of a Choosen Entry [**?**]

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 12: Catalogue View of a Choosen Entry [**?**]

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 13: 'MARC'-View of a Choosen Entry [**?**]

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 14: Screenshot of a Library Catalogue [**?**]

1.3.4 Central Geodata Warehouse

Recently it has become more common in city and canton governments to manage and update spatial data of all parts of the government in one central data pool called geodata warehouse (a spatial database management system). This makes huge savings as the data is now stored redundance free and needs to be updated only once. Until now, some data needed to be stored and updated in different departements of the government. This made it enormously difficult to have current data in all places. Additionally, when doing bigger projects the data can now simply be taken out of the geodata pool and does not need to be gathered in tedious and lengthy work.

The most important characteristics of such database systems:

Concurrent Use (Page 13)	A central geodata warehouse is a nice example of concurrent use of a database system. On the one hand different users get their data from it - the employees of the different departements. On the other hand different application software (e.g. GIS system) are used to access the geodata warehouse. Therefore, it is possible that the forestry departement uses GIS software A for access to the data while the surveyor departement uses GIS software B.
Separation of Data and Applications (Page 15)	As described above, different users' with different applications get access to the data. This is only possible, if the data is separated from the applications. If data were connected to the application it would be much work to process the data in way that other applications could read and use it. This independence is especially important in cases where the DBMS software needs replacement. With the separation of data and applications this is possible without having to re-write all of the application software. Example: A Microsoft Word file (where the data is included in the application format) is quite difficult to open with the Microsoft Excel software even though both applications are made by the same company.

Data Persistence (Page 18)	The capture of *geodata*: and other data is most often a lot of work and very expensive. Therefore, data persistence is a very important characteristic of a geodata warehouse. This way, it is possible to ensure that data is not lost and needs then to be replaced, which is costly.
Data Integrity (Page 16)	Governmental data often give information about legal conditions for example, the cadaster. Therefore, these data need to be thoroughly correct and reliable. That is achieved through the definition and following of specific consistency requirements and rules.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 15: Schematic Representation of a Geodata Warehouse and the possible Access to it from different Offices

1.3.5 Exercise

Find another database application (your work environment, internet,...) and try to figure out which are the most important characteristics of that application.

Write a short summary of this information (like the ones you have seen in this unit) and post your writing to the discussion board under the topic 'Database Applications'. Look at and discuss also at the postings from other students.

A tutor will comment on your summary (also posted to the discussion board).

1.4 Advantages and Disadvantages

From the file-card box to the database

Management and storage of data has changed a great deal over the years - from the file-card box, via the first (file based) digital version, to the modern database systems. The first part of this unit deals with file based systems in comparison to database systems. Afterwards, the advantages and disadvantages of database systems are discussed.

1.4.1 Comparison DBS versus file based

Knowing about the characteristics of a database system (unit Characteristics of the Database Approach (Page 12) and unit Example Applications (Page 20)) we will have a look at file based systems. With a file based approach each user defines and creates with a specific software the files he needs for a specific application. In comparison to the database system approach this results in some limitations.

Move the mouse over the bold terms to the left and the limitations of the file based approach in comparison to the database approach are explained in writing (right) and graphics (below).

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 16: Database Systems versus File Based Systems

Using a file based system it is possible that, for example, the administration of a school maintains information about the registration and fee payment of the students (e.g. based on a spreadsheet software). Meanwhile the lecturers manage data about the students and their grades. Even though both user groups are interested in student data they both might have different files and different software to update and change these. Such *redundancy* in the definition and storage of data wastes storage space. Additionally, the work needed to update data in more than one system is multiplied by the number of systems. In case above a change of basic student data like the address might need an update in the system of the lecturer and in the system of the administration as well.

1.4.2 Advantages of a DBMS

Basically, all in the unit Characteristics of the Database Approach (Page 12) listed features can be listed as advantages here too.

- Concurrent Use (Page 13)
- Structured and Described Data (Page 14)
- Separation of Data and Applications (Page 15)
- Data Integrity (Page 16)
- Transactions (Page 17)
- Data Persistence (Page 18)
- Data Views (Page 19)

Additionally, there are some advantages which were not yet explicitely mentioned.

Use the blue buttons to navigate.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 17: vorteile.swf

Click on the following link to get a list of the further advantages. (DOC file of the advantages (www.gitta.info/IntroToDBS/en/multimedia/furtherAdvantagesDBMS.doc))

1.4.3 Disadvantages of a DBMS

Beside the numerous advantages of a database system the disadvantages should not be kept secret.

Use the blue buttons to navigate.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 18: nachteile.swf

Click on the following link to get a list of the disadvantages. (DOC file of the disadvantages (www.gitta.info/IntroToDBS/en/multimedia/DisadvantagesDBMS.doc))

1.5 Tasks

The following task should allow you to test if you have understood the content of this lesson.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 19: tabellenkalkulationVsDbms.swf

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 20: zentralesGeodataWarehouse.swf

Drag the numbers beside the terms onto the circles in the graphic. Place them as exactly as possible and click '>Check' when you have placed them all. Symbols will show you which numbers are placed correctly and which are on the wrong place. The '>Reset' button clears the graphic from all numbers.

Attention: The '>Check' button only works correctly if all circles in the graphic are covered with a number.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 21: dragDrop_test_informationssystemaufbau.swf

1.6 Summary

In this lesson the significance of the data management based on database systems and the role of a database system within a geoinformation system (GIS) was explained. At the beginning, terms like database system and geoinformation system were defined and the terms data and information positioned in context.

Following important reasons for the use of database systems for the management of data in general and geodata in particular were given and illustrated with example applications of different fields of use. Included in the significant characteristics of the database approach are concurrent use as well as structured and described data. Additionally the advantages of the separation of data and applications and the features for high reliablitiy and data security like the concepts of transactions and data views were discussed.

1.7 Recommended Reading

- ZEHNDER, C.A., 1998 *Informationssysteme und Datenbanken* [**?**, Introduction into Information Systems and Databases, in German]

Glossary

Conceptual: A concept is an abstract, universal idea, notion, or entity that serves to designate a category or class of entities, events, or relations. Concepts are abstract in that they omit the differences of the things in their extension, treating them as if they were identical. They are universal in that they apply equally to everything in their extension. Concepts are also the basic elements of propositions, much the same way a word is the basic semantic element of a sentence.

Data (especially computer data): The presentation of facts, information or concepts which are created in a computer readable form or are translated into such a form.

Data and Database: An amount of data which are viewed by the operator as somehow going together plus additional data which is used by the DBMS to work correctly are called a database.

Database Management System: A database management system (DBMS) is a software product for the persistent, consistent and application independent storage and management of data. But also for the flexible and easy use of big and concurrently used databases.

Database System: A database system (DBS) consists of a DBMS and one or more databases.

Dimensions of Information: IIn the information theory three dimensions of information are distinguished: the syntactic, the semantic and the pragmatic. Let us take the example of a traffic light. In the syntactic dimension we differentiate the three colours red, yellow and green. But the traffic light makes more sense in the semantic dimension. In this dimension the colours are linked to meanings. Red means stop, green means go. However, only in the pragmatic dimension does the traffic light become useable for the traffic. Pragmatically, red means that the driver of a car must stop. [**?**]

Geodata: Geodata or data with a spatial relation are data about objects that through a position in space directly or indirectly can be referenced. The space is defined through a coordinate system which is in relation to the earth's surface.

Geoinformation System: "*A geoinformation system allows capturing, storing, analysing and presenting of all data that describe a part of the earth's surface and all on this part located technical and administrative equipment but also geoscientific, economic and ecologic features. (Translation)*" [**?**]

Information: Information is a useable answer to a concrete question. [**?**]

Information System: An information system extends the database with a couple of software tools for querying, presenting, transforming and analysing the data.

Metadata: Metadata is literally "data about data", is information that describes another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: It is data about the data in the book referred to by the card. Other common contents of metadata include the source or author of the described dataset, how it should be accessed, and its limitations. Another important type of data about data is the links or relationship among data.

Redundancy: Redundancy, in general terms, refers to the quality or state of being redundant, that is: exceeding what is necessary or normal, containing an excess. This can have a negative connotation, superfluous, but also positive, serving as a duplicate for preventing failure of an entire system.

Structured Data: Data is called structured if it can be subdivided systematically and linked.

Syntax: Syntax can in linguistics be described as the study of the rules, or "patterned relations" that govern the way the words in a sentence come together.

2 Database Systems: Concepts and Architectures

Knowing about the advantages of database supported data management and the huge possibilities of applying database systems this lesson will explain the basic concepts and typical architectures of database systems.

Firstly, it is discussed how a *conceptual*[7] scheme can be transferred into a database environment and which typical database models can be used for this task. Afterwards, the generic database architecture is explained, which should allow for the understanding the important aspects of the co-operation between the different schemes and the interfaces for the communication with a database management system.

Learning Objectives

- You know the relationship between data schemes, database models and database instances and are able to describe them.
- You are able to sketch and explain the 3-Scheme-Architecture.
- You know the meaning and principles of a database interface and are able to list their typical features.

[7] A concept is an abstract, universal idea, notion, or entity that serves to designate a category or class of entities, events, or relations. Concepts are abstract in that they omit the differences of the things in their extension, treating them as if they were identical. They are universal in that they apply equally to every thing in their extension. Concepts are also the basic elements of propositions, much the same way a word is the basic semantic element of a sentence.

2.1 Database Models, Schemes and Instances

With the help of databases facts and processes from the real world should be described and stored in digital form. The abstraction from the real world to the digital format is done with the help of models, so called database models.

In this unit, the most often used database models are presented. Following, database schemes are introduced as a formal description of a concrete database and their database instances or their database state.

2.1.1 Database Models

Database systems can be based on different data models or database models respectively. A data model is a collection of concepts and rules for the description of the structure of the database. Structure of the database means the data types, the constraints and the relationships for the description or storage of data respectively.

The most often used data models are:

Network Model and Hierarchical Model	The network model and the hierarchical model are the predecessors of the relational model. They build upon individual data sets and are able to express hierarchical or network like structures of the real world.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 22: Network Model and Hierarchical Model

Relational Model	The relational model is the best known and in today's DBMS most often implemented database model. It defines a database as a collection of tables (relations) which contain all data. This module deals predominantly with the relational database model and the database systems based on it.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 23: Relational Database Model

Object-oriented Model	Object-oriented models define a database as a collection of objects with features and methods. A detailed discussion of object-oriented databases follows in an advanced module.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 24: Schematic Representation of a Object-oriented Database Model

Object-relational Model	Object-oriented models are very powerful but also quite complex. With the relatively new object-relational database model is the wide spread and simple relational database model extended by some basic object-oriented concepts. These allow us to work with the widely know relational database model but also have some advantages of the object-oriented model without its complexity.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 25: Schematic Represenation of the object-relational Database Model

2.1.2 Database Schemes and Database Instances

Independent from the database model it is important to differentiate between the description of the database and the database itself. The description of the database is called **database scheme** or also *metadata*[8]. The database scheme is defined during the database design process and changes very rarely afterwards.

The actual content of the database, the data, changes often over the years. A database state at a specific time defined through the currently existing content and relationship and their attributes is called a **database instance**

The following illustration shows that a database scheme could be looked at like a template or building plan for one or several database instances.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 26: Analogy Database Schemes and Building Plans

When designing a database it is differentiated between two levels of abstraction and their respective data schemes, the conceptual and the logical data scheme.

Conceptual Data Scheme: : A conceptual data scheme is a system independent data description. That means that it is independent from the database or computer systems used. (Translated) [**?**]

Logical Data Scheme: A logical data scheme describes the data in a data definition language DDL of a specific database management system. (Translated) [**?**]

A logical data scheme describes the data in a data definition language DDL of a specific database management system. (Translated) (ZEHNDER 1998)

The conceptual data scheme orients itself exclusively by the database application and therefore by the real world. It does not consider any data

[8]Metadata is literally "data about data", is information that describes another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: It is data about the data in the book referred to by the card. Other common contents of metadata include the source or author of the described dataset, how it should be accessed, and its limitations. Another important type of data about data is the links or relationship among data.

technical infrastructure like DBMS or computer systems, which are eventually employed. *Entity relationship diagrams*[9] and relations are tools for the development of a conceptual scheme.

When designing a database the conceptual data scheme is derived from the logical data scheme (see unit Relational Database Design (www.gitta.info/LogicModelin/en/)). This derivation results in a logical data scheme for one specific application and one specific DBMS. A DB-Development System converts then the logical scheme directly into instructions for the DBMS.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 27: Schematic Represenation of the Different Schemes

[9] An entity is something that has a distinct, separate existence, though it need not be a material existence. In a relational database an entity is represented as a relation.

2.1.3 Comparison of Spatial Models and Database Models

Knowing about data modelling, the concepts and numerical models for the representation of spatial phenomena should be familiar.

The following comparison should make the differentiation between spatial models and database models a bit easier.

Spatial Models

Spatial models allow the modelling or representation respectively spatial phenomena of the real world like facts and processes. This happens firstly on a conceptual level, which means that, in general, the actual implementation of these models is not considered. Spatial models are often graphic oriented models like maps or plans and can be represented digitally or analogously (e.g. plaster model).

Examples for spatial models:

- The vector model is a special case of the object model.
- The raster model is a special case of the tesseral spatial model.

Database Models

Database models are from the category of informatic models and are therefore exact models or implementation models respectively. Database models can be used quite often and are not restricted to spatial types of problems.

Examples for database models:

- The relational database model.
- The object-relational database model.

In the intermediate module data management, the representation of spatial data models in database models and the representation of spatial data in geodatabase system will be discussed.

2.2 DBMS-Architecture and Data Independence

Database management systems are complex softwares which were often developed and optimised over years. From the view of the user, however, most of them have a quite similar basic architecture. The discussion of this basic architecture shall help to understand the connection with data modelling and the introductionally to this module postulated 'data independence' of the database approach.

2.2.1 Three-Schemes Architecture

Knowing about the conceptual and the derived logical scheme (discussed in unit Database Models, Schemes and Instances (Page 39) this unit explains two additional schemes - the external scheme and the internal scheme - which help to understand the DBMS architecture.

External Scheme: An external data scheme describes the information about the user view of specific users (single users and user groups) and the specific methods and constraints connected with this information. (Translated) [**?**]

Internal Scheme: The internal data scheme describes the content of the data and the required service functionality which is used for the operation of the DBMS. (Translated) [**?**]

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical scheme with data technical aspects like storage methods or help functions for more efficiency.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 28: Three-Schemes Architecture

The right hand side of the representation above is also called the three-schemes architecture: internal, logical and external scheme.

While the internal scheme describes the physical grouping of the data and the use of the storage space, the logical scheme (derived from the conceptual scheme) describes the basic construction of the data structure. The external scheme of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application. Therefore, a database has exactly one internal and one logical scheme but may have several external schemes for several applications using this database.

The aim of the three-schemes architecture is the separation of the user applications from the physical database, the stored data. Physically the data is only existent on the internal level while other forms of representation are calculated or derived respectively if needed. The DBMS has the task to realise this representation between each of these levels.

2.2.2 Data Independence

With knowledge about the three-schemes architecture the term data independence can be explained as followed: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

Physical Independence: Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimisation or reorganisation.

Logical Independence: Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

2.3 Database Languages and Database Interfaces

So far, we have got to know about data models, data descriptions and the components of a database system.

In this unit, it is explained how 'a data models gets into a database system' and 'how the information gets to the users'. More correctly formulated the following questions will be answered:

- How does an application interact with a database management system?
- How does a user look at a database system?
- How can a user query a database system and view the results in his/her application?

2.3.1 Database Languages

DDL

For describing data and data structures a suitable description tool, a **data definition language** (DDL), is needed. With this help a data scheme can be defined and also changed later.

Typical DDL operations (with their respective keywords in the structured query language SQL (www.gitta.info/RelQueryLang/en/)):

- Creation of tables and definition of attributes (CREATE TABLE ...)
- Change of tables by adding or deleting attributes (ALTER TABLE ...)
- Deletion of whole table including content (!) (DROP TABLE ...)

DML

Additionally a language for the descriptions of the operations with data like store, search, read, change, etc. the so-called data manipulation, is needed. Such operations can be done with a **data manipulation language** (DML). Within such languages keywords like insert, modify, update, delete, select, etc. are common.

Typical DML operations (with their respective keywords in the structured query language SQL (www.gitta.info/RelQueryLang/en/)):

- Add data (INSERT)
- Change data (UPDATE)
- Delete data (DELETE)
- Query data (SELECT)

Often these two languages for the definition and manipulation of databases are combined in one comprehensive language. A good example is the structured query language SQL which is discussed in detail in lesson Structured Query Language SQL (www.gitta.info/RelQueryLang/en/).

2.3.2 Database Interfaces

Working Principle of a Database Interface

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 29: Working Principle of a Database Interface

The application poses with the help of SQL, a query language, a query to the database system. There, the corresponding answer (result set) is prepared and also with the help of SQL given back to the application. This communication can take place interactively or be embedded into another language.

Type and Use of the Database Interface

Following, two important uses of a database interface like SQL are listed:

Interactive	SQL can be used interactively from a terminal.
Embedded	SQL can be embedded into another language (host language) which might be used to create a database application.

2.3.3 User Interfaces

A user interface is the view of a database interface that is seen by the user. User interfaces are often graphical or at least partly graphical (GUI - graphical user interface) constructed and offer tools which make the interaction with the database easier.

Form-based Interfaces

This interface consist of forms which are adapted to the user. He/She can fill in all of the fields and make new entries to the database or only some of the fields to query the other ones. But some operations might be restricted by the application.

Form-based user interfaces are wide spread and are a very important means of interacting with a DBMS. They are easy to use and have the advantage that the user does not need special knowledge about database languages like SQL.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 30: Example of a Form-based User Interface

Text-based Interfaces

To be able to administrate the database or for other professional users there are possibilities to communicate with the DBMS directly in the query language (in code form) via a input/output window.

We will see this possibility later in the lesson Structured Query Language SQL (www.gitta.info/RelQueryLang/en/).

Text-based interfaces are very powerful tools and allow a comprehensive interaction with a DBMS. However, the use of these is based on active knowledge of the respective database language.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 31: Example of a Text-base User Interface

GIS Interface

A GIS user interface often integrates features of a database interface. The database interaction takes place through the combination of different interfaces:

- Graphical interaction via a selection on the map
- Combination of form-based and text-based interaction (e.g. special Query-Wizards for the easier creation of database queries)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 32: Example of a GIS Interface (GeoMedia, Intergraph)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 33: Example of a Query-Wizard within a GIS

2.4 Tasks

Term matching

Drag the keywords below and drop them into the empty rectangles so that each time a pair of matching terms are formed. Click the check button to see if you did it correctly. The reset button sets all the terms back to their original position.

Hint: The terms in this exercise are used in the way they were introduced in this lesson. Literature is likely to use these terms slightly differently.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 34: wortzuordnung_einfach.swf

3-Scheme-Architecture

Drag the blue numbers near the keywords onto the correct white circles in the graphic. Click the check button to see if you did it correctly. The reset button sets all the numbers back to their original position.

Hint: The check button only works correctly after covering all of the empty circles in the graphic with a number.

Tip: Start with easy/known keywords.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 35: drag_drop_architektur_mod.swf

2.5 Exercise Data Independence

Describe in your own words the architecture of a database management system and try to answer the following question. What is data independence and how can it be ensured? If possible, add one or two examples from your knowledge which shows the advantage of data independence.

Your discussion should not be longer than 200-400 words. Post it to the discussion board under the topic 'Exercise Data Independence'. Read and discuss the postings of fellow students too.

2.6 Summary

In this lesson, the most often used database models (relational, object oriented and object relational) and the database technologies based on them were discussed. Such a database model offers some basic rules how a conceptual schema can be transfered in a specific database environment and described by means of a logical scheme. The data description is the base for the following creation and management of the database instances with the actual data. Logical modelling on the base of a relational data model will be discussed in the later lesson Logical Modelling (www.gitta.info/LogicModelin/en/).

Following the generic database architecture based on the 3-scheme-architecture was discussed. This 3-scheme-architecture extends the logical scheme with an internal and an external scheme. The concept of dividing it into three parts is an important prerequisite for the earlier mentioned data independence and the realisation of interfaces for the communication between applications and the database management system. Most of the known database systems offer an interface on the bases of the structured query language SQL. The basic elements and the use of SQL will be discussed in the later lesson Structured Query Language SQL (www.gitta.info/RelQueryLang/en/).

2.7 Recommended Reading

- ZEHNDER, C.A., 1998 *Informationssysteme und Datenbanken* [**?**, Introduction into information systems and databases, including data modelling, in German]

Glossary

Conceptional Data Scheme: A conceptual data scheme is a system independent data description. That means that it is independent from the database or computer systems used. (Translated) [**?**]

Conceptual: A concept is an abstract, universal idea, notion, or entity that serves to designate a category or class of entities, events, or relations. Concepts are abstract in that they omit the differences of the things in their extension, treating them as if they were identical. They are universal in that they apply equally to every thing in their extension. Concepts are also the basic elements of propositions, much the same way a word is the basic semantic element of a sentence.

Entity: An entity is something that has a distinct, separate existence, though it need not be a material existence. In a relational database an entity is represented as a relation.

External Scheme: An external data scheme describes the information about the user view of specific users (single users and user groups) and the specific methods and constraints connected with this information. (Translated) [**?**]

Internal Scheme: The internal data scheme describes the content of the data and the required service functionality which is used for the operation of the DBMS. (Translated) [**?**]

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical scheme with data technical aspects like storage methods or help functions for more efficiency.

Logical Data Scheme: A logical data scheme describes the data in a data definition language DDL of a specific database management system. (Translated) [**?**]

Logical Independence: Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

Metadata: Metadata is literally "data about data", is information that describes another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: It is data about the data in the book referred to by the card. Other common contents of metadata include the source or author of the described dataset, how it should be accessed, and its limitations.

Another important type of data about data is the links or relationship among data.

Physical Independence: Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimisation or reorganisation.

3 The relational database model

The relational database model is used in most of today's commercial databases. It is used since the early 80ies and was developed 1970 by E. F. Codd. The relational database model is based on a mathematical concept where relations are interpreted as tables.

The focus of this lesson lies in the conversion of a conceptual into a logical data scheme (the relational database model) using an entity-relationship-schema. You will find more information about schemas in the lesson about Database models, schemas and instances (www.gitta.info/DBSysConcept/en/).

3.1 Concept of the relational model

In contrast to the entity-relationship-model (ERM) which is a conceptual model, the relational model is a logical data model. It can be seen as lying one step or layer below the ERM. The relational model is not about abstract objects but defines how data should be represented in a specific database management system. The goal of a logical data model is to arrange the data in such a form that it is consistent, non-redundant and supports operations for data manipulation.

3.1.1 Data organization in a relational data model

A logical data schema (model) is in most cases based on a conceptual data scheme which, with the use of certain guidelines and rules, is transformed into a relational scheme (model). The main organization unit in a relational data model is the relation. A relation can be represented as a table but the definition of the relation is not necessarily equal to the definition of the table and vica versa.

Why use the relational model?

- **Simplicity.** Data in a relational model is represented through values that are structured with only one construct: the "relation".
- **Classification.** The relational model is based on mathematical fundamentals: the set theory.

3.1.2 Definitions

domain: A domain **D** is a set of atomic values that defines the value range of attributes.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 36: Example domain

tupel: A tuple **t** is a list with n values **t = <d1, d2, ..., dn>** where each value di is either an element of the domain **Di** or NULL. A tuple is a record in a relation (row in a table).

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 37: Example tupel

Attribute: A column of a table represents an attribute. It can also be described as a that a domain D has in arelation scheme R.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 38: Example attribute

Relation scheme: A relation scheme **R** (A1, A2, ..., An) is made up of a relation name R and a list of attributes A1, A2, ..., An.

Relation: A relation **r** is one instance of the relation scheme R(A1, A2, ..., An) containing a set of n-tuples r =t1, t2, ... , tn.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 39: Example relation scheme

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 40: Example relation

Relational database scheme: A relational database scheme is a set of relation schemes S = R1, ..., Rn together with a set of integrity conditions. A relational database is a relational database scheme together with a database instance.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 41: Relational scheme terms

The relational scheme of an object (entity) can be represented as a table (relation). In this example the entity are grades. This entity is described with the attributes name, subject and the grade. The domain (or value range) for the attributes name and subjects are all lower- and upper-case characters of the alphabet, the domain for the grades are real numbers from 1 to 6. The structure of this entity without any content is called a relational scheme. Any value that is entered has to be within the defined value range or domain. A row in the table is also called a tuple.

There is a small mistake in the above table. Do you find it?

3.2 Transforming an ERM to a relational database scheme

In this unit we will learn the rules and methods to represent entity relationship models in relational database schemes.

Each of the following mapping rules describes one of the components of the entity relationship model. To reduce an ERM into a relational scheme all 8 rules have to be worked out. Each rule has to be applied on every entity set (rule 1, 2, 7 and 8) or relationship set (rule 3, 4, 5 and 6). Usually the correct processing sequence is: 1, 7, 8, 2, 3, 4, 5, and then 6.

For every rule a definition and an example is given.

3.2.1 ERM concepts

Entity relationship diagram

To understand the following mapping rules, you must be familiar with the concepts of the entity relationship model (ERM). Using this flash example, you can refresh your memory about ERM.

hould any of the concepts be new, please consult the following book:

- Fundamentals of Database Systems, Elmasri, Ramez; Navathe, Shamakant B. (1994)

You should be familiar with the following terms: strong entity, weak entity, relationship, identifying relationship, attribute, derived attribute, multivalued attribute, composite attribute, identifier or primary key and discriminator.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 42: ERM concept basics

3.2.2 Rule 1

In this first step, all strong entity sets are transformed into the relational database schema. For subclasses use rule 8.

Definition rule 1

For each strong entity set G define a relational scheme R with the entity properties as attributes A. For multivalued attributes use rule 7. Define the primary key (identifier).

In this example you can see the application of rule 1::

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 43: Rule 1

A strong entity set is taken for the creation of the relation:

Customer

All entity properties are used as attributes for the relation:

Customer (CustomerNo, FirstName, LastName, Street, StreetNo)

The primary key (or identifier) is defined. Here it is the attribute CustomerNo:

Customer (CustomerNo , FirstName, LastName, Street, StreetNo)

3.2.3 Rule 2

In this step the weak entity sets are transformed into the relational database scheme.

Definition rule 2

For each weak entity set S with owner G create a relational scheme R with the entity properties as attributes A. For multivalued attributes use rule 7. Use the primary key of G as foreign key in R. Choose a discriminator (combination of attributes) that, together with the foreign key, will act as primary key for R. Note that only the combination of the foreign key together with the discriminator can act as primary key for R.

IIn this example you can see the application of rule 2:

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 44: Rule 2

A weak entity set is taken for the creation of the relation:

Part

All entity properties are used as attributes for the relation:

Part(Name, Editor)

The primary key of the owner's relational scheme is taken and added as a foreign key to the relation. In this example the owner is Newspaper(Name, Circulation, Price) and the primary key is Name which we use in the relation as NewspaperName (since Part already has a Name):

Part(NewspaperName, Name, Editor)

The discriminator attribute Name (of Part) together with the foreign key NewspaperName form the primary key of the relation Part:

Part(NewspaperName, Name , Editor)

3.2.4 Rule 3

In this step the binary relationships of the following type (1,1)(1,1), (0,1)(1,1) or (0,1)(0,1) are transformed into the relational database scheme.

Definition rule 3

Search for all binary relationships B (1,1)(1,1), (0,1)(1,1) or (0,1)(0,1). Find the relational scheme S and T that are connected through relationship B. Choose one of them (eg. S) and insert the primary of the other relational scheme (eg. T) as a foreign key. Also add the properties of B as attributes into this relational scheme.

In this example you can see the application of rule 3:

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 45: Rule 3

A binary relationship (type (1,1)(1,1)-, (0,1)(1,1)- and (0,1)(0,1)) is chosen: leads.

Then we choose an entity set that is related to this relationship:

Newspaper(Name , Circulation, Price)

The primary key of the second entity set (eg. ChiefEditor) is inserted as foreign key in the first relational scheme:

Newspaper(Name , Circulation, Price, *ChiefEditor_PersNo.)*

Now the properties of the relationship are used as attributes in the relational scheme:

Newspaper(Name , Circulation, Price, *ChiefEditor_PersNo.* , SinceDate)

3.2.5 Rule 4

In this step all binary relationships one to many/many to one (1,n)(1,1), (0,n) (1,1), (1,n)(0,1) or (0,n)(0,1) are transformed into the relational database scheme.

Definition rule 4

Search for all regular binary relationships B of type (1,n)(1,1), (0,n)(1,1), (1,n)(0,1) and (0,n)(0,1) and for their relational schemes S and T of the corresponding entity sets. Choose the relational scheme on the "(1,1)"/"(0,1)"-side (here S) and insert there the primary key of T as a foreign key. Also add the properties (if there are any) of B as attributes into this relational scheme.

In this example you can see the application of rule 4:

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 46: Rule 4

A binary relationship of type one to many/many to one ((1,n)(1,1)-, (0,n)(1,1)-, (1,n)(0,1)- und (0,n)(0,1)) is chosen: "commissions"

Then we choose the entity set that is on the "(1,1) or (0,1)"-side:

Advertisment(AdNo , Size, Price, PublDate)

The primary key of the second entity set (eg. Customer) is inserted as foreign key in the first relational scheme:

Advertisment(AdNo , Size, Price, PublDate, *ClientCustomerNo.)*

3.2.6 Rule 5

In this step all binary relationships many to many (0,n)(0,n), (1,n) (0,n) or (1,n)(1,n) are transformed into the relational database scheme.

Definition rule 5

Search for all regular binary relationships B of type many to many and the according relational schemes S and T. For each B create a new relational scheme R. The primary keys of S and T are used as foreign keys in R. Together they form the primary key of this new relational scheme R. Also add the properties (if there are any) of B as attributes into this relational scheme R.

In this example you can see the application

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 47: Rule 5

A binary relationship of type many to many ((0,n)(0,n)-, (0,n)(1,n)- and (1,n)(1,n)) is chosen: "published in"

For this relationship we define a new relational scheme:

AdpublishedInNewspaper

The primary keys of the connected relational schemes S and T are used as foreign key in the new scheme R. Together they form the primary key for R:

AdpublishedInNewspaper (Ad_OrderNo, Newspaper_Name)

Now the properties of the relationship are used as attributes in the relational scheme:

AdpublishedInNewspaper (NameAd_OrderNo, Newspaper_Name , Position)

3.2.7 Rule 6

In this step all relationships of order n are transformed into the relational database scheme. This happens according to rule 5.

Definition rule 6

For all n-ary relationship types (n>2) use rule 5: Create a new relational scheme R and use the primary keys of all connected relational schemes (S, T, etc.) as foreign keys. Together they form the primary key for this new relational scheme R.

See the example of rule 5.

3.2.8 Rule 7

If in rule 1 multivalued attributes are encountered, they are transformed into separate relational schemes according to this rule.

Definition rule 7

Define for each multivalued attribute A a new relational scheme R' which also contains this attribute A and the primary key of the according relational scheme R. Together they form the primary key for R'.

In this example you can see the application of rule 7:

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 48: RuleRegel 7

A new relational scheme is created for the multivalued attribute TelNo containing TelNo as attribute:

ChiefTelNo(TelNo)

The primary key of this multivalued attribute connected to the relational scheme is used as foreign key in the new scheme:

ChiefTelNo(*ChiefEditor_PersNo* , TelNo)

The foreign key and the attribute together form the primary key for this new relational scheme:

ChiefTelNo(ChiefEditor_PersNo, TelNo)

3.2.9 Rule 8

If in rule 1 you find subclasses then transform them according to this rule.

Definition rule 8

Define a relational scheme R for the superclass C with the attributes A(R)=(K, A1, A2, .., An), where K is the primary key. For each subclass create a new relational scheme Ri with their attributes and the primary key K of superclass C as an additional attribute. The primary key of Si is also K.

In this example you can see the application of rule 8:

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 49: Rule 8a

Using the superclass define a relational scheme R with a primary key K:

Employee(ENumber)

For each subclass create a relational scheme:

Technician

Engineer(Training)

Add the primary key K of the superclass as attribute of the subclasses and use it also as primary key:

Employee(ENumber),

Technician(ENumber),

Engineer(ENumber , Training)

3.2.10 Using the 8 rules

The following flash animation allows you to practice the 8 rules that you have now learned. Please click on the linked flash-graphic below and follow the instructions.

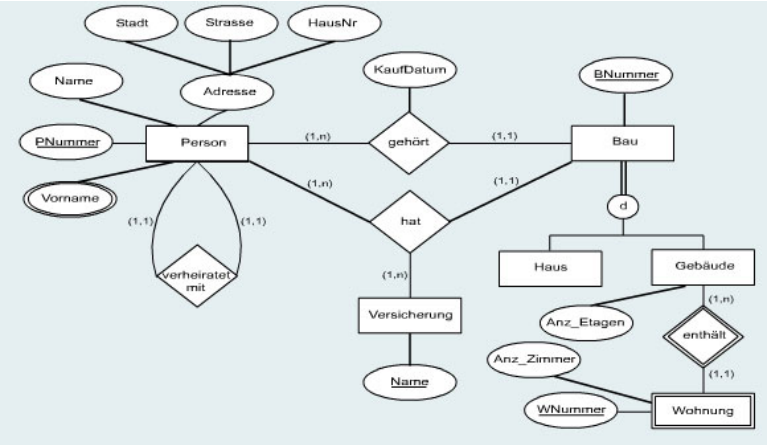

Figure 50: Flash exercise

3.2.11 Reducing an ERM to a relational scheme

Create a relational database scheme using the following conceptional data scheme (ERM - Entity Relationship Model) using the rules you here learned in this unit. Primary keys should be marked as underlined and foreign keys italic.

Publish your solution as a Word- or PDF-document on the discussion board. Check out the solutions of the other students and comment them. Any problems with this exercise can also be published on the discussion board. The solutions and questions will be checked by a tutor and general feedback published.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 51: A conceptional model (ERM)

3.3 Data integrity

Integrity or consistency stands for the quality and reliability of data of a database system. A database is consistent if the data reflects the referenced objects correctly. It is inconsistent if there exist ambiguous or contradictory tuples, relations or tables in the database.

3.3.1 Key integrity

Relations are defined as a unique set of tuples where no two tuples have the same combination of values for all their attributes. To be distinct, they must have some primary key that allows to reference each tuple. Key constraints deal with this issue.

Candidate key: Each attribute or minimal combination of attributes that uniquely identifies any tuple in a relation is called a candidate key. Minimal means that removing an attribute leaves the key without the ability to uniquely identify any tuple and therefore not being a candidate key anymore.

Primary key: The primary key is one chosen key candidate that acts as the identification key for a relation. Usually this is a short attribute like a ID-number (identification key) or username. Attributes of the primary are commonly underlined.

3.3.2 Entity integrity

The entity integrity is a result of the key integrity: No primary key is allowed to be the NULL(=no value). If NULL-values would be allowed for primary keys, than two tuples could have NULL as key value and therefore could not be distinguishably anymore. The key integrity would not be respected. In SQL special routines (unique, not null) are used for key and entity integrity.

Example:

ID	**Name**	**Surname**	**Year**
NULL	Miller	John	1955
NULL	Miller	John	1985

Table 1: Legend missing

Since the primary key (here ID) of both tuples is NULL, we are not able to distinct these tuples.

3.3.3 Referntial integrity

In a relationship model, in contradiction to an entity relationship model ERM, there is no way to model relationships between tuples explicitly. Therefore relationships are modeled implicitly using a primary- and foreign-key concept.

Foreign key: An attribute in a relational scheme R1 is a foreign key if it is in relationship with a primary key from R2 and if:

- The domain (value range) of the foreign key in R1 is the same as the domain of the primary key in R2.
- The set of values of the foreign key in R1 is a subset of all primary key values in R2.

Foreign keys usually are marked dotted underlined.

A relationship between two schemes is established by using the domain of the primary key in one scheme as the domain of a foreign key in a second scheme (with the according attributes).

Referential integrity constraints ensure that any foreign key value is always pointing on an primary key of an existing tuple. References to non-existing primary keys are not allowed. In such a case the foreign key value must be set to NULL.

The relational scheme containing the foreign key is called the referencing scheme, the scheme with the primary key is called the referenced scheme.A foreign key can also point to its own scheme.

In the early days of computer databases, only workstations were able to check for referential integrity. Today most PC-based database systems are able to check for referential integrity.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 52: Example Referntial integrity

In the above table Division we have Div_No as a primary key. In the second table Employee this key is used as a foreign key to associate each member to one department. For each department number in the table Employee there exists a department in the table Division. These tables are integer. If we

would insert a tuple "4, Weber, A5" into table Employee then a database system should refuse this operation because in table Division there exists no value A5 and therefore there is no integrity.

3.3.4 Integrity endangering operations

There are three types of operation that could potentially endanger referential integrity:

- Inserting of a new tuple
- Deleting an existing tuple
- Updating attribute values of existing tuples

Of course with all these operations there has to be a check on all integrity rules. Selecting values (browsing in the database) does not change any values and is therefore no integrity endangering operation.

Inserting of a new tuple

With this operation all three integrity rules are concerned. The following problems can occur:

- Key integrity: The for this insertion used primary key value is already used in another tuple.
- Entity integrity: The primary key of the new tuple is NULL.
- Referential integrity: A foreign key without an associated primary key is used.

These operations must either be rejected by the database system or transformed into consistent operations according to defined rules.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 53: Examlpe Inserting of tuples

Deleting an existing tuple

With this operation only referential integrity is concerned. If a foreign key points on a primary key that has been deleted, then the foreign key becomes invalid. The database system should react on a delete operation with one

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 54: Example Deleting existing tuples

of the following solutions: Abort the operation, cascading delete (also the referenced tuple is deleted!) or set the value of the foreign key to NULL.

By deleting the tupel Div_No "A1" in the relation Division, the division number of the tupel ID "1" in the relation Employee would become invalid.

Updating attribute values of existing tuples

With this operation all three integrity rules are concerned. An update operation can be seen as first a delete of an existing tuple followed by a insert of a new tuple with updated values. Therefore all the above rules must be applied. Problems arise only with the updating of primary or foreign keys. All other attributes can be updated without any restriction.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 55: Example Updating attribute values of existing tuples

By modifying the value Div_No in the relation Division from "A1" to "A4", the division number of ID "1" in the relation Employee would become invalid.

3.4 Normalization

Normalisationis a process which we analyze and alter a database relation in order to get more concise and organized data structures. Normalised data is stable and has a natural structure. We call a relation normalized if:

- it does not contain any redundancy
- it does not cause maintenance problems
- it is an accurate representation of the data

Relations that aren't normalised contain non-atomic attributes and therefore can contain redundant information. Detailed planning of the ERM can help creating normalised relations. The following steps will explain how existing relations can be normalised step by step.

3.4.1 Dependencies

In order to be able to normalise a relation according to the three normal forms, we must first understand the concept of dependency between attributes within a relation.

Functional dependency: If A and B are attributes of relation R, B is functionally dependenton A (denoted A –> B), if each value of A in R is associated with exactly one value of B in R.

Example:

ID	**Name**
S1	Meier
S2	Weber

Table 2: Legend missing

The attribute Name is functionally dependent of attribute ID (ID –> Name).

Identification key: If every attribute B of R is functionally dependent of A, than attribute A is a primary key.

Beispiel:

ID	**Name**	**Surname**
S1	Meier	Hans
S2	Weber	Ueli

Table 3: Legend missing

Attribute ID is the identification key

Full functional dependency: We talk about full functional dependency if attribute B is functional dependent on A, if A is a composite primary key and B is not already functional dependent on parts of A.

Beispiel:

IDStudent	**Name**	**IDProfessor**	**Grade**
S1	Meier	P2	5
S2	Weber	P1	6

Table 4: Legend missing

The attribute Grade is fully functional dependent on the attributes IDStudent and IDProfessor.

Transitive dependency: If A determines B and B determines C then C is determined by (dependent on) A. We write A –> B and B –> C but not B –> A.

Example:

ID	**Name**	**Konto_Nr**	**Bank_Code_No**	**Bank**
L1	Meier	1234-5	836	UBS
L2	Weber	5432-1	835	CS

Table 5: Legend missing

There is a transitive dependency between Bank_Code_No and Bank because Bank_Code_No is not the primary key of the relation.

3.4.2 First normal form (1NF)

First normal form: A relation is in first normal form if every attribute in every row can contain only one single (atomic) value.

A university uses the following relation:

Student(Surname, Name, Skills)

The attribute Skills can contain multiple values and therefore the relation is not in the first normal form.

But the attributes Name and Surname are atomic attributes that can contain only one value.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 56: Example First normal form

To get to the first normal form (1NF) we must create a separate tuple for each value of the multivalued attribute

3.4.3 Second normal form (2NF)

Second normal form: A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.

A university uses the following relation:

Student(IDSt, StudentName, IDProf, ProfessorName, Grade)

The attributes IDSt and IDProf are the identification keys.

All attributes a single valued (1NF).

The following functional dependencies exist:

1. The attribute ProfessorName is functionally dependent on attribute IDProf (IDProf –> ProfessorName)

2. The attribute StudentName is functionally dependent on IDSt (IDSt –> StudentName)

3. The attribute Grade is fully functional dependent on IDSt and IDProf (IDSt, IDProf –> Grade)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 57: Example Second normal form

The table in this example is in first normal form (1NF) since all attributes are single valued. But it is not yet in 2NF. If student 1 leaves university and the tuple is deleted, then we loose all information about professor Schmid, since this attribute is fully functional dependent on the primary key IDSt. To solve this problem, we must create a new table Professor with the attribute Professor (the name) and the key IDProf. The third table Grade is necessary for combining the two relations Student and Professor and to manage the grades. Besides the grade it contains only the two IDs of the student and the professor. If now a student is deleted, we do not loose the information about the professor.

3.4.4 Third normal form (3NF)

Third normal form: A relation is in third normal form if it is in 2NF and no non key attribute is transitively dependent on the primary key.

A bank uses the following relation:

Vendor(ID, Name, Account_No, Bank_Code_No, Bank)

The attribute ID is the identification key. All attributes are single valued (1NF). The table is also in 2NF.

The following dependencies exist:

1. Name, Account_No, Bank_Code_No are functionally dependent on ID (ID –> Name, Account_No, Bank_Code_No)

2. Bank is functionally dependent on Bank_Code_No (Bank_Code_No –> Bank)

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 58: Example Third normal form

The table in this example is in 1NF and in 2NF. But there is a transitive dependency between Bank_Code_No and Bank, because Bank_Code_No is not the primary key of this relation. To get to the third normal form (3NF), we have to put the bank name in a separate table together with the clearing number to identify it.

3.4.5 Exercise normalization

The following table is already in first normal form (1NF). There is only one entry per field. Please convert this table to the third normal form (3NF) using the techniques you learned in this Unit. Write a short report about your solution and post it in the discussion board. Check the other solutions and comment them, if necessary. If you have questions, you can also post them in the discussion board. A tutor will look at the questions regularly and give feedback and answers.

A table with the students and their grades in different topics.

UnitID	**StudentID**	**Date**	**TutorID**	**Topic**	**Room**	**Grade**	**Book**	**TutEmail**
U1	St1	23.02.03	Tut1	GMT	629	4.7	Deumlich	tut1@fhbb.ch
U2	St1	18.11.02	Tut3	GIn	631	5.1	Zehnder	tut3@fhbb.ch
U1	St4	23.02.03	Tut1	GMT	629	4.3	Deumlich	tut1@fhbb.ch
U5	St2	05.05.03	Tut3	PhF	632	4.9	Dümmlers	tut3@fhbb.ch
U4	St2	04.07.03	Tut5	AVQ	621	5.0	SwissTopo	tut5@fhbb.ch

Table 6: Legend missing

3.4.6 Unit summary

This Unit showed how and why relations should be normalised. It prevents problems and saves money. A careful planning in the conceptual phase helps implementing and normalising a table properly. Although there is a fourth and a fifth normal form (that were not discussed here), usually it is enough to normalise up to the third normal form.

3.5 Summary

In a relational model real world objects are represented in tables. Each table is made out of rows and columns. Each row, also known as tuple or record, is made out of fields, also known as attributes. Each Attribute stands for a certain feature of the real world object. An attribute is defined by a name and its value.

Relations between tuples represent existing relationships between objects (tables). Furthermore key attributes have to be defined (usually displayed underlined in a relation). They are necessary for the allocation (relation) of objects (tables) and allow unique accesses to tables.

Integrity or consistency stands for the quality and reliability of data of a database system. A database is consistent if the data reflects the referenced objects correctly. It is inconsistent if there exist ambiguous or contradictory tuples, relations or tables in the database.

A relation model (scheme, entity) should reflect relationships that also logically (in the real world) belong together. To avoid anomalies different types of normalisations help keeping the database consistent.

3.6 Recommended Reading

- ELMASRI, R., NAVATHE, S.B., 1994 *Fundamentals of Database Systems* [**?**, Introduction in databases and SQL (in English).]

Glossary

Attribute: A column of a table represents an attribute. It can also be described as a that a domain D has in arelation scheme R.

Candidate key: Each attribute or minimal combination of attributes that uniquely identifies any tuple in a relation is called a candidate key. Minimal means that removing an attribute leaves the key without the ability to uniquely identify any tuple and therefore not being a candidate key anymore.

domain: A domain **D** is a set of atomic values that defines the value range of attributes.

First normal form: A relation is in first normal form if every attribute in every row can contain only one single (atomic) value.

Foreign key: An attribute in a relational scheme R1 is a foreign key if it is in relationship with a primary key from R2 and if:

- The domain (value range) of the foreign key in R1 is the same as the domain of the primary key in R2.
- The set of values of the foreign key in R1 is a subset of all primary key values in R2.

Foreign keys usually are marked dotted underlined.

Full functional dependency: We talk about full functional dependency if attribute B is functional dependent on A, if A is a composite primary key and B is not already functional dependent on parts of A.

Functional dependency: If A and B are attributes of relation R, B is functionally dependenton A (denoted A –> B), if each value of A in R is associated with exactly one value of B in R.

Identification key: If every attribute B of R is functionally dependent of A, than attribute A is a primary key.

Primary key: The primary key is one chosen key candidate that acts as the identification key for a relation. Usually this is a short attribute like a ID-number (identification key) or username. Attributes of the primary are commonly underlined.

Relation: A relation **r** is one instance of the relation scheme R(A1, A2, ..., An) containing a set of n-tuples r =t1, t2, ... , tn.

Relational database scheme: A relational database scheme is a set of relation schemes S = R1, ..., Rn together with a set of integrity conditions. A relational database is a relational database scheme together with a database instance.

Relation scheme: A relation scheme **R** (A1, A2, ..., An) is made up of a relation name R and a list of attributes A1, A2, ..., An.

Second normal form: A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.

Third normal form: A relation is in third normal form if it is in 2NF and no non key attribute is transitively dependent on the primary key.

Transitive dependency: If A determines B and B determines C then C is determined by (dependent on) A. We write A –> B and B –> C but not B –> A.

Tupel: A tuple **t** is a list with n values **t = <d1, d2, ..., dn>** where each value di is either an element of the domain **Di** or NULL. A tuple is a record in a relation (row in a table).

4 Structured Query Language SQL

SQL (Structured Query Language) is a query language for relational databases. The roots of SQL go back to SQUARE, a more mathematical oriented language and SEQUEL, a predecessor of SQL from the seventies. There exist different standards (ISO and ANSI) of SQL, but the most common one is SQL-92. The latest standard SQL-99 (or SQL3) even includes XML. In the following Units we will give a simple overview of the most important SQL functions.

This lesson is geared to the SQL-99 standard [**?**] . However, this standard is not implemented by all database systems constantly. For example, some commands are named differently or their syntax is constructed differently. For this reason, SQL commands from this lesson might not work in certain systems. If you encounter problems, please check in the respective user manuals whether or not the command is supported at all, whether it is named, or constructed differently. Most database systems also go beyond the standard. However, caution is advised with these SQL extensions since they can vary considerably from system to system.

Learning Objectives

- You understand the basic SQL concepts and are able to explain the use of SQL in the areas of data definition, data manipulation, and data control.
- You are able to use SQL to create, modify, and delete tables.
- You can express simple and complex queries with the help of SQL and know how SQL queries are used in nesting or in combinations. You are also able to correctly apply arithmetic operators and set operators within SQL queries.
- You master SQL to add, modify, and delete data tuples.

4.1 SQL overview

SQL (Structured Query Language) is one of the main reasons for the commercial success of relational databases. The ANSI (American National Standards Institute) and the ISO (International Standards Organization) developed in 1986 the first SQL-version with the name SQL-86 or SQL1. In 1992 a second and more extended standard with the name of SQL-92 or SQL2 was established. The latest standard includes XML, dates from 1999 and is therefore called SQL-99 or SQL3. With the use of SQL in most commercial database systems the migration from one system to another has become easier for the user. In the ideal case, the user need not consider which system is used because query formation in SQL remains the same.

4.1.1 SQL Concepts

SQL is a descriptive, entity-oriented query language for data manipulation with its roots in relational algebra. Today SQL is used either as a stand-alone programming language or within other languages like C, C++, Java, ADA, COBOL, FORTRAN, PL/1, PASCAL etc.

SQL actually consists of three sub languages:

- DDL - Data Definition Language: Used for creating databases and tables and for maintaining the structure.
- DML - Data Manipulation Language: Used for accessing and extracting data from a database (add, update, delete etc.).
- DCL - Data Control Language: Used to control access to the database and therefore essential for the security system.

In most implementations of SQL functions from other programming languages (if-clauses, iterations etc.) have been added. Some SQL-versions, such as Oracle's PL/SQL, can therefore be seen as independent programming languages.

4.1.2 Data Definition (DDL)

In SQL the terms table, row and column are synonyms for relation, tuple and attribute. To create a new relation scheme in the database we start with the `create table` -command. Together with the creation we must provide the relations attributes and their domains (e.g. number, char or date). Additional we can define other constraints, checks and keys etc. The keyword `NOT NULL` tells the system that this attribute cannot be empty. Primary keys are declared using a special `''table constraint''-clause` .

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 59: Code-Beispiel: Create Table

This example shows how a new table is created in SQL. In the intermediate lesson we take a closer look at all the SQL statements. For the moment you don't have to understand these statements in detail.

4.1.3 Data Manipulation (DML)

There are two sorts of data manipulation commands in SQL. The first type are only used for database queries and do not alter the tables. The second type are used for adding, updating or deleting values in a database.

We will take a closer look at the different data manipulation commands in the following units.

- Basic database queries (Page 108)
- SQL Insert, Delete and Update (Page 125)

4.1.4 Data control (DCL)

Data control commands in SQL control access privileges and security issues of a database system or parts of it. These commands are closely related to the DBMS (Database Management System) and can therefore vary in different SQL implementations.

Some typical commands are:

- `GRANT` - give user access privileges to a database
- `DENY` - deny user access
- `REVOKE` withdraws access privileges given with the GRANT or taken with the DENY command

Since these commands depend on the actual database management system (DBMS), we will not cover DCL in this module.

4.2 Creation and modification of tables

In this unit, we will show you how to create, modify, and delete tables using SQL commands.

4.2.1 Create tables

With `CREATE TABLE` a new table can be created in a database. The command has the following basic structure:

`CREATE TABLE <table name> (<Attribute definitions and constraints>);`

The table name must be unique within the current database or the current scheme.

Attribute definition

Attributes are defined by a name and a datatype, whereas the name must be unique within the table. These specifications are compulsory for all attributes.

The order of the attributes at definition corresponds to the order of the columns in the table created. If a certain order is aspired, you need to define it at the creation of the table. Unless there are no changes made to the table (see Change table structure (Page 105)), the order stays this way.

Constraints

There are two types of constraints: table constraints and attribute constraints. The difference is that attribute restrictions apply to only one attribute whereas table constraints may apply to more than one attribute but this need not be. With these restrictions, the range of values of the attributes can be restricted or it is prevented that the entered values are not allowed. A record cannot be recorded if it violates a restriction.

There are four kinds of constraints:

- `UNIQUE` - the attribute or the attribute combination need to be unique within the table
- `PRIMARY KEY` - the attribute or the attribute combination is the primary key of the table
- `FOREIGN KEY` - the attribute is a foreign key
- `CHECK` - Condition that must be fulfilled for an attribute or an attribute combination

The constraints can be named. However, this is not necessary.

In this example, a table is added to a database. That table is linked to an already existing table. The difference between an attribute and a table constraint can be seen in the SQL command. projekt_ID and leiter_ID have an

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 60: CREATE TABLE Befehl

attribute constraint (the constraint is written directly behind the attribute definition). projekt_ID has the constraint PRIMARY KEY - it is therefore the primary key of this table, i.e. the attribute must be unique and must not be NULL. leiter_ID has the constraint NOT NULL (special case of a CHEK constraint), meaning it needs to hold a value at all times.

The link to the existing table is defined as a table constraint (FOREIGN KEY) and is named (projektleiter). This constraint could also be defined as an attribute constraint since it only includes one attribute.

The example shows that there are basically no difference between attribute and table constraints as long as only one attribute is affected. It is about two different ways of collecting constraints.

4.2.2 Changing the table structure

With `ALTER TABLE` the structure of a table can be modified. The attributes and constraints that were created with CREATE TABLE can be modified, new ones can be added, and existing ones can be deleted. The command has the following syntax:

`ALTER TABLE <table name> <Change> ;`

whereas `<Change>` can include various commands:

- `ADD [COLUMN] <Attribute defintion>`
  Add an attribute (Attribute definiton as in CREATE)
- `ALTER [COLUMN] <Attribute name> SET DEFAULT <Standard value>`
  define a new standard value
- `ALTER [COLUMN] <Attribute name> DROP DEFAULT`
  delete current standard value
- `DROP [COLUMN] <Attribute name> RESTRICT | CASCADE`
  delete an attribute
- `ADD <Table constraint>`
  add new table constraint (table constraint as in CREATE)
- `DROP CONSTRAINT <Table constraint>`
  delet a table constraint

With the above commands, attributes and constraints can be added or deleted respectively. In addition, standard values for the attributes can be set or deleted. There are other SQL commands that are not listed here.

Default SQL does not include any commands for modification or renaming of attributes. This would lead to problems when data already exists. However, in some databases these commands are included (e.g. MODIFY or RENAME). The syntax is different in every system though. If there are no data, the attribute to be changed can be delete and reattached.

In this example, an attribute is added to a table. The dataset shown contains NULL for this attribute because there was no value assigned yet. Afterwards, this attribute is delted from the table again. The keyword

`RESTRICT` provokes that only attributes that are not linked to other tables can be deleted (foreign key). Alternatively, the keyword

`CASCADE` can be used. Using this, not only the designated column but also the linked column in the other table is delted.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 61: ALTER TABLE command

4.2.3 Deleting tables

With DROP TABLE an existing table can be deleted. The data and structure of the table are deleted.

The command has the following syntax:

```
DROP TABLE <Table name>;
```

Attention: With `DROP TABLE` the table structure is deleted with all the data. In most cases, this cannot be undone!

4.3 Basic database queries

In this Unit we will take a closer look at how to do database queries using SQL.

4.3.1 SELECT-FROM-WHERE clause

In languages like SQL data from a certain domain (FROM) that match some conditions (WHERE) are selected and presented (SELECT). The result can be seen as a new relation.

The syntax of a basic SQL query is:

`SELECT <select-list>`

`FROM <from-list>`

`WHERE <condition>`

In this syntax the...

- <select-list> contains the names of the attributes (columns) values to be returned.
- <from-list> is the list of the relations (tables) used for the query.
- <condition> is an expression that identifies the tuples that we are looking for.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 62: Basic SQL queries

We saw the basic statements that are needed for making queries with SQL. Of course there are extensions which allow more specific or more flexible queries.

These extensions include:

- multiple conditions (in boolean AND/OR combination)
- complex conditions (subqueries, joins etc.)
- pattern matching and arithmetical operators
- non-relational functions (sort, group, aggregate)
- set operators (union, intersect, minus)

4.3.2 Multiple conditions

In SQL it is possible to have multiple conditions and combine them with boolean operators (AND, OR). For negating a condition the NOT operator is used. The data is selected if the whole condition returns as TRUE.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 63: Multiple conditions

Other examples of multiple conditions:

- `PLZ = 8000 AND NOT Last name = 'Schmidt'`
- `PLZ = 8000 OR PLZ = 8006`
- `(Last Name = 'Müller' OR Last name = 'Meier') AND Place = 'Zürich'`

In SQL queries (e.g. conditions) text must be set in single quotation marks. Numerical values are written without single quotation marks. This can be seen in the examples.

Instead of a long chaining with AND there is also the possibility to compare several attributes simultaneously with each other:

`First name = 'Ursula' AND Last name = 'Müller'` newLine space="long" />

can be written as:

`(First name, Last name) = ('Ursula', 'Müller')`

4.3.3 Comparison operators

SQL supports different comparison operators. They can be used to compare attributes with constants or with other attributes. DIf an operator is used to compare an attribute with a constant, we talk about restriction. If the operator is used to compare two attributes, then we talk about a join. With joins, data from different relations can be compared and combined. Of course only attributes with the same domain (value range) can be compared.

The following comparison operators for numerical attributes are supported:

- `=` equal
- `>` greater
- `<` less
- `>=` greater than or equal
- `<=` less than or equal
- `<>` not equal
- `BETWEEN` between

Conditions usually have the syntax `<atribute> <operator> <value>` . The operator `BETWEEN` has a somehow different syntax (see the example).

Other comparison operators are discussed in the next paragraphs.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 64: Comparison operators

If in the SELECT-clause a star (*) instead of an attribute list is used, then all attributes of the relations specified in the FROM part of the query are displayed.

4.3.4 Pattern matching and arithmetical operators

Pattern matching using LIKE

The LIKE condition allows you to use wildcards in the WHERE clause of an SQL statement. This allows pattern matching.

The patterns that you can choose from are:

- "%" allows you to match any string of any length (including zero length)
- "_" allows you to match on a single character

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 65: Pattern matching using LIKE

`NOT LIKE` examines whether the given section is not present in the string.

Arithmetical operators

The arithmetical standard operators for addition (+), subtraction (-), multiplication (*) and division (/) can all be used for numerical constant or for attributes with a numerical domain. This allows attributes in a request to be calculated together.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 66: Arithmetical operations query

In the above example the result is a relation with three attributes. The third attribute is named "Usury" since "Price/Size" is not a good name and stands for the price per unit. We define that if it lies over 15, it is overpriced ("Usury").

Arithmetical operators can be used in the SELECT part of a request or in the conditions.

In the SELECT or FROM part of a request, attributes and relations can be given a different name by the command `AS : <Attribute/Relation> AS <new name>` (The key word `AS` can be omitted). This can be used to give a calculated value a meaningful name (see example), or to make SQL requests easier to read.

4.3.5 Nested queries

Nested queries

Conditions are usually made of an attribute, a value and an operator that forms the condition (eg. Name="John"). But the values themselves don't have to be constants, they can be the result of another sub-query. We then talk about nested queries. Using the IN-operator nested queries can be as deep as necessary.

There are three types of sub-queries that differ in their result:

- Sub-queries that return a value (one column and one row)
- Sub-queries that return a row
- Sub-queries that return several rows

If only one value or row is returned the normal comparison operators can be used.

If more than one row is returned special operators are used:

- `IN` examines whether the value exists in the sub-query
- `<Comparison operator> ALL` the condition must return TRUE for all rows in the sub-query
- `<Comparison operator> ANY (SOME)` the condition must return TRUE for at least one row in the sub-query

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 67: Nested queries

Sub-queries can also be used in the FROM part of a query. Thus, a relation can be compiled specifically for a request. The sub-query must be assigned a name with AS: `(<Sub-query>) AS <Name>`

4.3.6 Join

It happens often that data from multiple realtions are required in a query. For this, relations need to be linked. The identical attributes (foreign keys) in both relations are linked.

There are two ways to link relations in queries:

in the WHERE part

In this option, the identical attributes are linked with the comparison operator and inserted as "condition" (not a real condition) into the WHERE part of the query.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 68: Join operators

Basically, e.g. a < can also be used for a join. However, this is usually used in combination with a join condition with = since such a join alone does not make sense.

in the FROM part

Another possibility is to do the join in the FROM part of a query. This makes more sense because the actual search criteria (conditions in the WHERE part) are separated from the table joins.

The following commands are available for this purpose:

- `<Relation> JOIN <Relation> USING (<Attribute>)`

  the relations are joined by an attribute with the same name in both tables

- `<Relation> NATURAL JOIN <Relation>`

  automatically joins all attributes that are of the same name in both relations.

- `<Relation> JOIN <Relation> ON <Attribute> <comparison operator> <Attribute>`

  with this command, you can decide by which attributes the relations should be joined and which operator is to be used (there can be more than one join as well).

Using these commands, the example from above would look like the following:

```
SELECT name, surname, newspaper_name
FROM customer JOIN subscription USING (CustNo);
```

or

```
SELECT name, surname, newspaper_name
FROM customer JOIN subscription ON customer.CustNo = subscription.CustNo;
```

or

```
SELECT name, surname, newspaper_name
FROM customer NATURAL JOIN subscription;
```

The commands above only return datasets present in both relations. Should **all** datasets of a relation be returned with the corresponding datasets of the second relation, the following commands are applied:

- `<Table> RIGHT OUTER JOIN <Table> USING (<Attribute>)`

  all datasets of the right relation and the corresponding datasets of the left relation

- `<Table> LEFT OUTER JOIN <Table> USING (<Attribute>)`

  all datasets of the left relation and the corresponding datasets of the right relation

If there are no joins to be made in the left relation using `RIGHT OUTER JOIN` , NULL is returned for the attributes of this relation. The same is true for the opposite (using `LEFT OUTER JOIN`).

4.3.7 Non-relational constructs

ORDER BY clause

SQL contains some operators that have nothing to do with relational algebra. For example an entity per definition does not have an order. Nevertheless in SQL you can order your tables using the ORDER BY clause. Using the keywords ASC and DESC the sorting can either be ascending or descending. If no keyword is used, the sorting is in ascending order by default.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 69: ORDER BY clause

In this example all customers are sorted in ascending order according to their names and as a second sort parameter in descending order according to their surname. The ASC keyword is default and could therefore be omitted.

GROUP BY clause

Grouping methods are used to form a subset of tuples of a relation according to certain criteria. These subsets can then be used to calculate statistical parameters such as average, sum etc. The value of a certain attribute serves as grouping criteria. All tuples with the same value for this attribute are grouped. These groups can be used in further processes (a special case would be another grouping to be able to use groups of groups). For this, so-called group functions are used. They can only be applied to numerical attributes.

The group functions that SQL usually offers are the following:

- `min` returns the smallest value ignoring null values
- `max` returns the largest value ignoring null values
- `sum` returns the sum of all values ignoring null values
- `count` returns the number of rows
- `avg` returns average value ignoring null values
- `stdev` returns the standard deviation ignoring null values
- `varriance` returns the variance ignoring null values

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 70: GROUP BY clause

In this query we want to find the customers that spent more than 250 SFR for all their small advertisings (a small add is an add that costs less than 300 SFR). In a first step restriction is applied: A004 is sorted out because it is not a small add (price is over 300 SFR). The remaining tuples are grouped by customer and those with a sum of over 250 SFR are selected (customer groups 002 adn 005 are sorted out because they have not reached the limit yet).

Requests including a GROUP BY clause are processed as follows: First, the condition in the WHERE part is processed (if applicable). Then, the specified columns are grouped. With the condition in the HAVING part of the request, another condition can, if necessary, be specified for the grouped attribute values. Due to this order, you can see that there cannot be a group function in the condition of the WHERE part of the request since there has been no grouping yet. However, all attributes appearing in the request must be either in the GROUP BY clause or in a group function.

4.3.8 Set operators

Set operators are used to connect different queries and produce a resulting relation. Of course these set operations are only allowed if the attributes match (eg. the domain or the number of attributes etc.). The following set operators are used to join together sets of tuples. They might already be known from the algebra of sets.

- `union` produces a union of different sets
- `intersect` produces an intersection of different sets.
- `minus` subtracts one set from another one.

By default, duplicates are not returned using a request with these set operators. If duplicates need to be returned, the keyword `ALL` must be put behind `UNION` , `INTERSECT` or `EXCEPT` .

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 71: Set operators

This query is used to find all names of customers who have a magazine subscription and who have placed an advertising in this magazine.

4.3.9 Summary

In this Unit it was shown step by step how SQL statements are phrased. To show all the parts in context a complete SQL request is listed below.

Complete syntax of a SQL request:

```
SELECT [DISTINCT | ALL]
<Attribute> [AS <Name>] [, ...]  | *
FROM <Relation> [, <Relation>]
[WHERE <Condition>]
[GROUP BY <Column> [HAVING <Condition>]
[ORDER BY <Column> [ASC | DESC], [, ...]];
```

A large part of this syntax is optional and only needs to be stated only in certain cases. Teh simplest syntactically correct request has only a SELECT- and a FROM-part. Only with more complex problems, all parts are used in the same request.

It is possible that certain datasets appear more than once in the result of a request. The keyword `DISTINCT` is used to delete such duplicates from the result. The keyword `ALL` is used to prevent the duplicates from deletion. This is the default and does not have to be stated explicitly.

4.3.10 Database queries

This self assessment should give you the possibility to test different SQL queries and get to know them more in depth. We use the following sample database (www.gitta.info/RelQueryLang/en/multimedia/Datenbasis.html) (please leave this overview open during the exercise).

Find the correct solution tables for the following SQL queries. Put your solutions (either Word or PDF format) on the discussion board and check the other solutions.

You are encouraged to comment on other postings or to ask questions using the discussion board. Please do not email the tutor directly so that others can also benefit from your questions and the tutors answer.

SQL Requests

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 72: sql_I.gif

-

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 73: sql_II.gif

-

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 74: sql_III.gif

-

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 75: sql_IV.gif

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 76: sql_V.gif

-
-

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 77: sql_VI.gif

-

Solutions to the exercises

download solutions (www.gitta.info/RelQueryLang/en/download/Loesung_SQLAnfrage.pdf)Filesize: 183KB. Type: pdf.

This self assessment uses the same sample database (www.gitta.info/RelQueryLang/en/multimedia/Datenbasis.html)as before but now using an interactive SQL tool called WebSQL. (captiva.ifi.unizh.ch:8085/websql/) This tool was developed by the GITTA-partner IFI (Institute of informatics, University of Zurich). Please contact your tutor if you didn't get your account information or if the login doesn't work.

Some remarks about the WebSQL interface

- The semicolon at the end of a SQL query or SQL command is optional.

- On the tab 'Dataset' you find information about the content of the original database.
- You changes to the database are saved and are available at your next login.
- On the tab 'Settings' you can change your user information and reset the database to its original state. All your changes will be deleted. Note that this command is irreversible.
- On the tab 'Protocol' you find a list of all your SQL queries.
- Please contact your tutor and not the IFI for technical questions.

The goal of this self assessment is that you are able to form SQL queries that answer the questions below. Write a report where you explain your queries and comment on the solutions and put this document on the discussion board. If you have any questions please post them also on the discussion board. Of course you are welcome to check the other students solutions, comment on them or answer questions from other students.

Aufgaben

- Select the names of all employees and sort them in descending order (Z to A).
- Find the names of all employees who live in Dübendorf and earn more than 30'000 Fr.
- Select all children whose parents are employees and live in Zurich. Sort the result by birth date.
- Find the total amount of time that is spent on all projects in the research (Forschung) department. Use one query only!
- How many children has the manager of the administration (Verwaltung) department?
- Find projects of Zurich where the total amount of work is over 50 hours?
- Create your own SQL query and post the query (in word form) together with the solution on the discussion board (under the topic 'eLSQL').
- Solve some queries that other students have posted.

Possible solutions

download solutions (www.gitta.info/RelQueryLang/en/download/Loesung_eLSQL1.pdf)Filesize: 179KB. Type: pdf.

4.4 SQL Insert, Delete and Update

To keep a database accurate we have to be able to not only create and delete tables but to also modify the content. In SQL this is done with the commands *INSERT*, *DELETE* and *UPDATE*.

4.4.1 Inserting tuples

In its most basic form the INSERT INTO command adds a tuple to an existing table.

The syntax is:

`INSERT INTO` <Tablename> (<Attribute list>)

`VALUES` (<Valuelist>);

Please note that the attribute list can be omitted if a complete tuple is inserted.

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 78: Inserting tuples

With the INSERT command shown above, new tuples are inserted into the database. Often, the values to be inserted are the result of a request. For this situation the command can be changed as follows:

`INSERT INTO <Tablename> (<Attribute list>)`

`SELECT ...` (normal request)

VALUES is replaced by an SQL request. The result of this request is inserted into specified table. If the request returns a complete tuple in the correct order the attribute list does not need to be stated here either.

4.4.2 Deleting tuples

The DELETE FROM command removes one or more rows (tuples) of a table (relation). The WHERE-clause specifies which tuples have to be deleted. If it is missing, all tuples are deleted!

The syntax is:

`DELETE FROM` <Tablename>

`WHERE` <Condition>;

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 79: Deleting tuples

NOTE: If `DELETE FROM` is used without `WHERE` , all datasets of this table are deleted. This is irreversible in most systems.

4.4.3 Updating tuples

The UPDATE command is used to change one or more attribute values of existing tuples. The WHERE-clause specifies which tuples should be updated. The SET-clause specifies which attributes should be changed and their new values.

The syntax is:

`UPDATE` <Tablename>

`SET` Attributevalues

`WHERE` <Condition>;

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 80: Updating tuples

The assignment of the new value happens as follows:

`<Attributname> = <Value>`

Multiple assingments are separated by commas. The assigned value can either be a constant (absolute term) or the result of a request, i.e. there can be a request to the right of the equals sign.

If a whole row is updated, the assignment looks as follows:

`ROW = ROW(<list of values>)`

4.4.4 eLSQL Exercise 'Insert, Delete und Update'

This exercise predominantly addresses the SQL commands INSERT, DELETE, and UPDATE. To complete the exercise however, the SQL requests from unit "Basic database queries" need to be known as well. During the following exercise you will work with this sample database (www.gitta.info/RelQueryLang/en/multimedia/Datenbasis.html). (please leave this overview open during the exercise). You can also find the overview in the eLSQL Tool on the tab 'Dataset'. The exercise is executed through a web interface on a MySQL database. Your tutor will provide you the information needed (webaddress and registration).

Some remarks about eLSQL interface

- The semicolon at the end of a SQL query or SQL command is optional.
- On the tab 'Dataset' you find information about the content of the original database.
- You changes to the database are saved and are available at your next login.
- On the tab 'Settings' you can change your user information and reset the database to its original state. All your changes will be deleted. Note that this command is irreversible.
- On the tab 'Protocol' you find a list of all your SQL queries.
- Please contact your tutor and not the IFI for technical questions.

Exercises

- You have been hired by a company (represented in the sample database (www.gitta.info/RelQueryLang/en/multimedia/Datenbasis.html)) and have to add yourself in the table employee. Add at least someone of your family in the according table. Also you should add some hours of work to one of the projects.

  Remark: You don't have to add your birth date since the date format may be a bit complicated!
- Select the columns you added in the different tables.
- Project 20 has been finished. Please delete all entries related to project 20.

  How did you proceed and why?

- Beate Tell leaves the company. Her position is taken over by Sonja Maradona. Please make sure that these changes are made correctly in your company database. Delete all entries that are no longer needed.

  How did you proceed and why?

- All employees who's boss is Boris Frisch get a wage raise of 10'000 Fr.

- Present some other typical database changes (and their solutions) that might have to be done in this company.

Possible solutions

download solutions (www.gitta.info/RelQueryLang/en/download/Loesung_eLSQL2.pdf)Filesize: 265KB. Type: pdf.

4.5 Usage of SQL

This flash-animation should allow you to exercise all kinds of database queries.

To start the animation, klick on the graphic

Please note:
Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed.

Figure 81: Flash-Animation

The data of this exercise can be downloaded as Access-Datenbase (www.gitta.info/RelQueryLang/en/multimedia/SQL-SelfAss2.mdb)Filesize: 180 KB. Type: mdb. . All requests from the exercise can be tested to find the difference.

4.6 Summary

SQL stands for "Structured Query Language" and is a language to communicate with relational and object oriented databases. With SQL new tables (relations, schemes) can be created, altered, and deleted using the commands `CREATE TABLE` , `ALTERTABLE` and `DROP TABLE` . This part of SQL is also known as the Data Definition Language (DDL). More important in the daily use of SQL are the data query and manipulation (DML) commands. These commands allow you to `INSERT, DELETE, and UPDATE` values in the database. SQL is also used to control access restrictions to the database or to parts of it.

4.7 Recommended Reading

- ELMASRI, R.; NAVATHE, S.B., 1994 *Fundamentals of Database Systems* [**?**, Introduction to databases and SQL.]