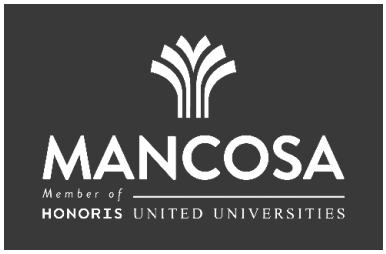
Bachelor of Commerce in Information and Technology Management (Year 2)

INFORMATICS 2A

Module Guide


MANCOSA
Member of
HONORIS UNITED UNIVERSITIES

Bachelor of Commerce
in Information and Technology Management (Year 2)
INFORMATICS 2A

Table of Contents

Preface

A. Welcome

Dear Student

It is a great pleasure to welcome you to **Informatics 2A (INF2A6)**. To make sure that you share our passion about this area of study, we encourage you to read this overview thoroughly. Refer to it as often as you need to, since it will certainly make studying this module a lot easier. The intention of this module is to develop both your confidence and proficiency in this module.

The field of **Informatics** is extremely dynamic and challenging. The learning content, activities and self- study questions contained in this guide will therefore provide you with opportunities to explore the latest developments in this field and help you to discover the field of **Informatics** as it is practiced today.

This is a distance-learning module. Since you do not have a tutor standing next to you while you study, you need to apply self-discipline. You will have the opportunity to collaborate with each other via social media tools. Your study skills will include self-direction and responsibility. However, you will gain a lot from the experience! These study skills will contribute to your life skills, which will help you to succeed in all areas of life.

Welcome to the MANCOSA Informatics 2A module.

We are living in the information age, and information is the currency of our times - the right information at the right time can be worth millions. Businesses and organizations with the most up-to-date and relevant information will almost certainly find themselves ahead of the competition in a global and highly competitive market. As a result, organizations that involve themselves with storing, sorting, aggregating and presenting information have become the leading businesses in the world; typical examples of these are Google and Facebook.

There is a lot of information available, and much more being produced every minute. This avalanche of information will only be useful if it is stored in such a way that it is readily available and easily accessible to those who require it, without delay. If the information is not stored correctly, it could easily become useless.

Much of the world's data is stored in powerful databases which are spread out globally and live on the internet and other networks. These databases are at work twenty four hours a day, constantly receiving, storing, sorting, aggregating and dispensing data to millions of users globally; they are truly a miracle of technology. It is these databases which ensure that the data available to individuals, businesses and organizations is always up-to-date, accurate and relevant.

It goes to reason then, that databases form the backbone of the information age. Databases, however, will only perform well if they have been designed well; a poorly designed database will not be able to produce the right information at the right time. As an information technology professional, it will hence be your responsibility to ensure that the databases which store valuable data belonging to your organization have been designed according to the rules of proper database design, as laid out by computer scientists.

This module will provide you with a sound understanding of the major concepts underlying database technology, and will enlighten you on the best practices in database design and implementation. Using the knowledge and skills you acquire from this module, you will be able to branch out into the vast, challenging yet lucrative field of database design and management.

We hope you enjoy the module.

B. Module Overview

- The module is a **15** credit module at NQF level **6**.

How to study for this Module

Distance learning is seldom the easiest option, and with a highly practical subject like Informatics 2A, it becomes that much more challenging, particularly for a complete newcomer to the topic. We at MANCOSA have put a many systems into place to ensure that you as the learner have everything you need to make your learning process as simple and as possible, starting with this study guide.

This study guide will serve as the basis of your studies through this course.

1. Commence by reading through a section carefully, in order to get a fair understanding of the concepts being discussed
2. Where references are made to sections in the prescribed text, read those sections as well
3. Read both texts a second, possibly a third time, getting a firm grasp on the concepts being discussed
4. Complete all the prescribed activities. Remember that these activities are not optional!
5. The 'Think point' and case study examples have been carefully selected to provide you with insights into the real world of software engineering. It is very important that you complete these as well.
6. At the end of the chapter, do the Review questions
7. Above all, persevere!

Your time allocation for this module should be no less than 8 hours per week.

This study guide comprises of 8 units:

- **Unit 1: Database systems**

  In this unit you will be introduced to the concept of databases and some of the reasons why databases are the best choice when it comes to information storage.

- **Unit 2: Data models**

  Like the architectural plans of a house, database models help to plan and conceptualize the database prior to building it, which is essential in order to create a well-designed and robust database. In this unit you will be taken through the first steps to database design by understanding the basic concepts of data modelling.

- **Unit 3: The relational database model**

  Relational databases are by far the most popular type of databases in the world today, due to a number of factors. As a result, you will almost certainly be working with relational databases should you enter the world of IT. By the end of this unit you will be familiar with some of the characteristics of relational databases, along with some major relational concepts.

- **Unit 4: Entity relationship modelling**

  In unit 2 you were introduced to the ideas behind data modelling. In this unit you will begin to create actual data models.

- **Unit 5: Normalization**

  Once you have designed your database, you will need to refine it until it reaches a state of perfection. The process of normalization can be seen as a refining and fine-tuning process for your database, and ensures that data which will be stored in the database will be free of errors and anomalies.

- **Unit 6: Introduction to Structured Query Language**

  Database management systems provide programmers with a simple, structured and universal language through which they can issue all kinds of instructions to the database. This language is known as the structured query language (SQL). In this unit you will learn SQL.

- **Unit 7: Transaction management and concurrency control**

  Modern databases, especially those which live on the web, are typically accessed by millions of users every second, and hence need to provide ways to ensure that there is no confusion and no errors leading to in the data anomalies despite the huge traffic. In this unit you will learn about transaction management and concurrency control, two powerful mechanisms provided by relational databases to handle multi-user environments.

- **Unit 8: Developing an object-orientated computer program in Java to manipulate a relational database**

  This unit provides a quick look into what is in store in Informatics 2B.

Module outcomes

- Purpose of this module:

  To ensure that the learner will have knowledge on theoretical aspects of database models (relational model), on the entity-relationship (E-R) modeling procedure and on the normalizing of a database's relations. To expose the learner to practical experience in the design of a relational database and the related E-R modelling and normalization

C. Exit Level Outcomes and Associated Assessment Criteria of the Programme

Exit Level Outcomes (ELOs)	Associated Assessment Criteria (AACs)
• The structure of data, information and knowledge in an organisational setting	• Structure of data, information and knowledge is reviewed within an organisational setting to emphasize data processes
• The application of information and knowledge management principles and theories in a variety of organisational settings	• Information and knowledge management principles and theories are applied in a variety of organisational settings to eliminate disarray
• The application of information and knowledge management principles in the different functional units of an enterprise	• Information and knowledge management principles is applied in different functional units of an enterprise to maintain consistency
• The architecture, platforms and configuration of systems to generate information and knowledge for decision making	• Architecture, platforms and configuration of systems is examined to generate information and knowledge for decision making
• Utilisation of information and knowledge management as a strategic tool for competitive advantage	• Information and knowledge management is applied as a strategic tool for competitive advantage to promote organisational information and knowledge management skills
• Analysis, evaluation and representation of financial, quantitative and functional information and knowledge for meaningful interpretation	• Financial, quantitative and functional information and knowledge is analysed, evaluated and presented for meaningful interpretation to establish organisational information and knowledge management needs

• Proposal of business solutions through information and knowledge management techniques	• Business solutions are proposed utilising information and knowledge management techniques to enhance organisational performance

D. Learning Outcomes and Associated Assessment Criteria of the Module

LEARNING OUTCOMES OF THE MODULE	ASSOCIATED ASSESSMENT CRITERIA OF THE MODULE
• Introduce the student to the fundamental concepts and principles of Information Communication Technologies (ICT).	• Fundamental concepts and principles of information Communication Technology (ICT) are introduced to aid in understanding databases
• Develop the student's understanding of the core building blocks of Information Technology in relation to hardware, software, communication and collaboration using the Internet, intranet or extranet.	• Hardware, software, communication and collaboration using the Internet, intranet or extranet are reviewed in understanding the building blocks of Information technology
• Provide the student with the foundation for understanding the impact of ICT's actions in the organisation's future.	• Actions related to the impact of ICT towards an organisations future is explored and forms the basis in understanding ICT
• Advance the student's insight into Enterprise Wide Information	• Enterprise wide information is elaborated to aid in understanding types of systems
• Technology and relate the benefits of Workflow and Business	• Technology associated with Work flow and business is reviewed to understand the related benefits to an organisation.
• Process Management to the organisation's future	• Process Management tools and techniques are examined within an organisation and aids in determining its future.

E. Learning Outcomes of the Units

You will find the Unit Learning Outcomes on the introductory pages of each Unit in the Module Guide. The Unit Learning Outcomes lists an overview of the areas you must demonstrate knowledge in and the practical skills you must be able to achieve at the end of each Unit lesson in the Module Guide.

F. Notional Learning Hours

Types of learning activities	Learning time
	%
Lectures/Workshops (face to face, limited or technologically mediated)	**15**
Tutorials: individual groups of 30 or less	**0**
Syndicate groups	**0**
Practical workplace experience (experiential learning/work-based learning etc.)	**0**
Independent self-study of standard texts and references (study guides, books, journal articles)	**27**
Independent self-study of specially prepared materials (case studies, multi-media, etc.)	**40**
Other: Online	**18**
TOTAL	**100**

G. How to Use this Module

This Module Guide was compiled to help you work through your units and textbook for this module, by breaking your studies into manageable parts. The Module Guide gives you extra theory and explanations where necessary, and so enables you to get the most from your module.

The purpose of the Module Guide is to allow you the opportunity to integrate the theoretical concepts from the prescribed textbook and recommended readings. We suggest that you briefly skim read through the entire guide to get an overview of its contents. At the beginning of each Unit, you will find a list of Learning Outcomes and Associated Assessment Criteria. This outlines the main points that you should understand when you have completed the Unit/s. Do not attempt to read and study everything at once. Each study session should be 90 minutes without a break

This module should be studied using the prescribed and recommended textbooks/readings and the relevant sections of this Module Guide. You must read about the topic that you intend to study in the appropriate section before you start reading the textbook in detail. Ensure that you make your own notes as you work through both the textbook and this module. In the event that you do not have the prescribed and recommended textbooks/readings, you must make use of any other source that deals with the sections in this module. If you want to do further reading, and want to obtain publications that were used as source documents when we wrote this guide, you should look at the reference list and the bibliography at the end of the Module Guide. In addition, at the end of each Unit there may be link to the PowerPoint presentation and other useful reading.

H. Study Material

The study material for this module includes tutorial letters, programme handbook, this Module Guide, a list of prescribed and recommended textbooks/readings which may be supplemented by additional readings.

I. Prescribed and Recommended Textbook/Readings

The prescribed and recommended readings/textbooks presents a tremendous amount of material in a simple, easy-to-learn format. You should read ahead during your course. Make a point of it to re-read the learning content in your module textbook. This will increase your retention of important concepts and skills. You may wish to read more widely than just the Module Guide and the prescribed and recommended textbooks/readings, the Bibliography and Reference list provides you with additional reading.

The prescribed and recommended textbooks/readings for this module is:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

- Morris Stephen et al, Database Principles – Fundamentals of Design, Implementation and Management 2nd Edition ISBN: 9781408066362

J. Special Features

In the Module Guide, you will find the following icons together with a description. These are designed to help you study. It is imperative that you work through them as they also provide guidelines for examination purposes.

Special Feature	Icon	Explanation
LEARNING OUTCOMES		The Learning Outcomes indicate aspects of the particular Unit you have to master.
ASSOCIATED ASSESSMENT CRITERIA		The Associated Assessment Criteria is the evaluation of the students' understanding which are aligned to the outcomes. The Associated Assessment Criteria sets the standard for the successful demonstration of the understanding of a concept or skill.
THINK POINT		A Think Point asks you to stop and think about an issue. Sometimes you are asked to apply a concept to your own experience or to think of an example.

ACTIVITY		You may come across Activities that ask you to carry out specific tasks. In most cases, there are no right or wrong answers to these activities. The purpose of the activities is to give you an opportunity to apply what you have learned.
READINGS		At this point, you should read the references supplied. If you are unable to acquire the suggested readings, then you are welcome to consult any current source that deals with the subject.
PRACTICAL APPLICATION OR EXAMPLES		Practical Application or Examples will be discussed to enhance understanding of this module.
KNOWLEDGE CHECK QUESTIONS		You may come across Knowledge Check Questions at the end of each Unit in the form of Knowledge Check Questions (KCQ's) that will test your knowledge. You should refer to the Module Guide or your textbook(s) for the answers.
REVISION QUESTIONS		You may come across Revision Questions that test your understanding of what you have learned so far. These may be attempted with the aid of your textbooks, journal articles and Module Guide.
CASE STUDY		Case Studies are included in different sections in this Module Guide. This activity provides students with the opportunity to apply theory to practice.
VIDEO ACTIVITY		You may come across links to Videos Activities as well as instructions on activities to attend to after watching the video.

Unit 1:

Database Systems

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
1.1 Why Databases?	• Understand the difference between data and information
1.2 Data vs. Information	• Describe what a database is, the various types of databases, and why they are valuable assets for decision making
1.3 Databases and DBMS's	• Understand the importance of database design
1.4 Role and Advantages of the DBMS	• Explain how modern databases evolved from file Systems
1.5 Types of databases	• Be aware of flaws in file system data management
1.6 Evolution of File System Data Processing	• Know the main components of the database system
1.7 The Database System Environment	• Understand the main functions of a database management system (DBMS)
1.8 Summary	• Summarise topic areas covered in unit

Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

1.1 Why Databases?

Databases contain data and information needed in businesses, big or small, in order to make decisions. Databases are collections of data that allow computer-based systems to store, manage and retrieve data very quickly. These databases help decision makers to know what they want to know when they want to know it.

Prescribed and Recommended Textbooks/Readings

Read section 1.1 of Rob et al to get a firm grasp of the concepts of data and information.

1.2 Data vs. Information

Rob, Coronel et al describe data as the raw facts that have not yet been processed to reveal their meaning. These raw data can be saved to a data repository. You can think of a repository as a recycle bin, where your deleted files are stored to first before finally deciding to delete the files from your PC completely and for good. These raw data can then be extracted, processed and transformed into a data summary for easy reading and review. Finally, for presentation, this data in the summary can be translated to visual forms like graphs, to enhance your ability to extract meaning from the data gathered at the very beginning.

Information is the result of processing raw data to reveal its meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modelling. To reveal its meaning, information requires context. Having a number is just raw data, but adding additional information like a metric like cm, or km, shows context, a meaning, to that number.

Raw data must be properly formatted for storage, processing and presentation. Timely and useful information requires accurate data. These data must be properly generated and stored in a format that is easy to access and process. Finally, the data environment must be managed carefully. Data management is a discipline that focuses on the proper generation, storage and retrieval of data.

To summarize:

- Data constitutes the building blocks of information
- Information is produced by processing data
- Information is used to reveal the meaning of data
- Accurate, relevant and timely information is the key to good decision making
- Good decision making is the key to organizational survival in a global environment

Transforming raw data into information – study figure 1.1

1.3 Databases and DBMS's

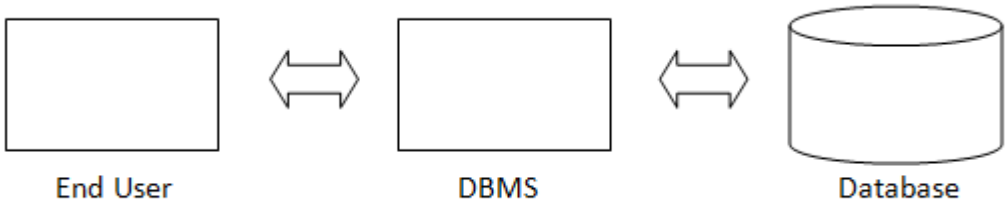
A database is a shared, integrated computer storage structure that stores:

1. Raw data
2. Metadata. Metadata is data about data. It provides a description of the data characteristics and the set of relationships that links the data found within the database. Metadata provides information that compliments and expands the value and use of the data.

A database management system (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database.

1.4 Role and Advantages of the DBMS

The Database Management System (DBMS) serves as the intermediary between the user and the database. The database structure is stored as a collection of files, and the only way to access the data in those files is through the DBMS.



The DBMS:

- Receives all application requests and translates them into the complex operations required to fulfil those requests.
- Hides much of the database's internal complexity from the application programs and users.

Some important advantages of the DBMS are:

- Enables the data in the database to be shared among multiple applications and users.
- Integrates the many different users' views of the data into a single all-encompassing data repository.

A DBMS provides other advantages, such as:

- Improved data sharing
- Improved data security
- Better data integration
- Minimized data inconsistency
- Improved data access
- Improved decision making
- Increased end-user productivity

Data inconsistencies occur when different versions of the same data appear in different places. The probability of data inconsistency is greatly reduced in a properly designed database.

1.5 Types of databases

A DBMS can support many different types of databases. There are numerous types of databases, and they can be classified according to the number of users:

- **Single-user database** supports only one user at a time. A single-user database that runs on a personal computer is called a desktop database.

- **Multiuser database** supports multiple users at the same time. It, in turn, has two types:
  - **Workgroup database:** When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called a workgroup database.

  - **Enterprise database:** When the database is used by the entire organization and supports many users (more than 50, usually hundreds) across many departments, the database is known as an **enterprise database**.

Databases can be classified according to its location.

- **Centralized database:** A database that supports data located at a single site is called a **centralized database**.

- **Distributed database**: A database that supports data distributed across several different sites is called a **distributed database**.

Databases are popularly classified according to the extent of use and on the time sensitivity of the information gathered from them.

- **Operational database**: A database that is designed primarily to support a company's day-to-day operations is classified as an **operational database**. This is also referred to as a **transactional** or **production database**.

- **Data warehouse:** In contrast to operational databases, a data warehouse focuses primarily on storing data used to generate information required to make tactical or strategic decisions.

Databases can also be classified to reflect the degree to which the data are structured.

Unstructured data are data that exist in their original (raw) state. **Structured** data are the result of taking unstructured data and formatting (structuring) such data to facilitate storage, use, and the generation of information. **Semi-structured** data are data that have already been processed to some extent. An **XML database** supports the storage and management of semi-structured XML data.

The table below shows some of the most popular database systems on the market, and what type they fall under:

Product	**Number of users**			**Data location**		**Data usage**	
	Single user	Mulituser-Workgroup	Multiuser-Enterprise	Centralized	Distributed	Operational	Data Warehouse
MS Access	X	X		X		X	
MS SQL Server	X	X	X	X	X	X	X
IBM DB2	X	X	X	X	X	X	X
Oracle RDBMS	X	X	X	X	X	X	X

Why is database design important?

Database design refers to the activities that focus on the design of the database structure that will be used to store and manage end-user data. Proper database design requires the designer to identify precisely the database's expected use. Designing a transactional database emphasizes accurate and consistent data and operational speed. Designing a data warehouse database emphasizes the use of historical and aggregated data. Designing a database to be used in a centralized, single-user environment requires a different approach from that used in the design of a distributed, multiuser database. Designing appropriate data repositories of integrated information using the two-dimensional table structures found in most databases is a process of decomposition. A well-designed database facilitates data management and generates accurate and valuable information.

1.6 Evolution of File System Data Processing

Manual File Systems

Historically, filing systems were often manual, paper-and-pencil systems. The papers within these systems were organized in an order which suited the usage of the data. For example, some files may have been stored in alphabetical order, while others in numerical order.

Computerized File Systems

With the advent of computers, files began to be stored digitally. Initially, the computer files within the file system were similar to the manual files. The description of computer files requires a specialized vocabulary to enable practitioners to communicate clearly. Some of the important terms relating to data storage are as follows:

- *Data:* These are raw facts that have little meaning unless they have been organized in some logical manner.
- *Field:* A character or a group of characters that has a specific meaning, used to define and store data.
- *Record:* This is a logically connected set of one or more fields that describes a person, place, or thing.
- *File:* This is a collection of related records.

Prescribed and Recommended Textbooks/Readings

Read section 1.4 of Rob et al to gain insights into the evolution of data storage systems.

Problems with File System Data Processing

The file system of storing information was a definite improvement on manual systems, but it had its own set of problems, such as:

- A file system requires extensive programming
- There are no ad-hoc query capabilities – in other words, each time a single entry is required from a file, the entire file would be loaded and read.
- System administration can be complex and difficult
- It is difficult to make changes to existing structures. Modifications are likely to produce errors, and additional time is spent using a debugging process to find those errors.
- Security is usually inadequate

These limitations of file systems severely restrict the type and nature of data that can be stored in them.

Structural and Data Dependence

Another major issue with file systems is that they exhibit **structural dependence** and **data dependence**, which are described as follows:

- **Structural dependence**: A file system exhibits **structural dependence** when the access to a file is dependent on its structure. Conversely, **structural in-dependence** exists when it is possible to make changes in the file structure without affecting the application program's ability to access the data.

- **Data dependence**: A file system exhibits **data dependence** when all data access programs are subject to change when any of the file's data storage characteristics change. Conversely, **data in-dependence** exists

when it is possible to make changes in the data storage characteristics without affecting the application program's ability to access the data.

The practical significance of data dependence is the difference between the **logical data format** (how the human being views the data) and the **physical data format** (how the computer must work with the data).

Data Redundancy

In a file system it is almost always inevitable to have the same data stored in different places, leading to islands of information and data redundancy.

- **Islands of information**: The term "**islands of information"** refers to data which is scattered in different locations. This is because the organizational structure of file systems promotes the storage of the same basic data in different locations.
- **Data redundancy:** Data redundancy exists when the same data are stored unnecessarily at different places. Data redundancy undoubtedly leads to serious problems such as:
- Poor data security: having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access.
- Data inconsistency: exists when different and conflicting versions of the same data appear in different places.
- Data anomalies: *anomaly* is defined as being an abnormality. In data redundancy, anomalies forces field value changes in many different locations. There are three types of data anomalies (Referring to the figure below):
- Update anomalies: For example, if agent Leah F. Hahn has a new phone number that number must be entered in each of the CUSTOMER file records in which Ms Hahn's phone number is shown.
- Insertion anomalies: For example, if only the CUSTOMER file existed, to add a new agent, you would also add a dummy customer data entry to reflect the new agent's addition.
- Deletion anomalies: For example, if you delete the customers Amy B. O'Brian, George Williams, and Olette K. Smith, you will also delete John T. Okon's agent data.

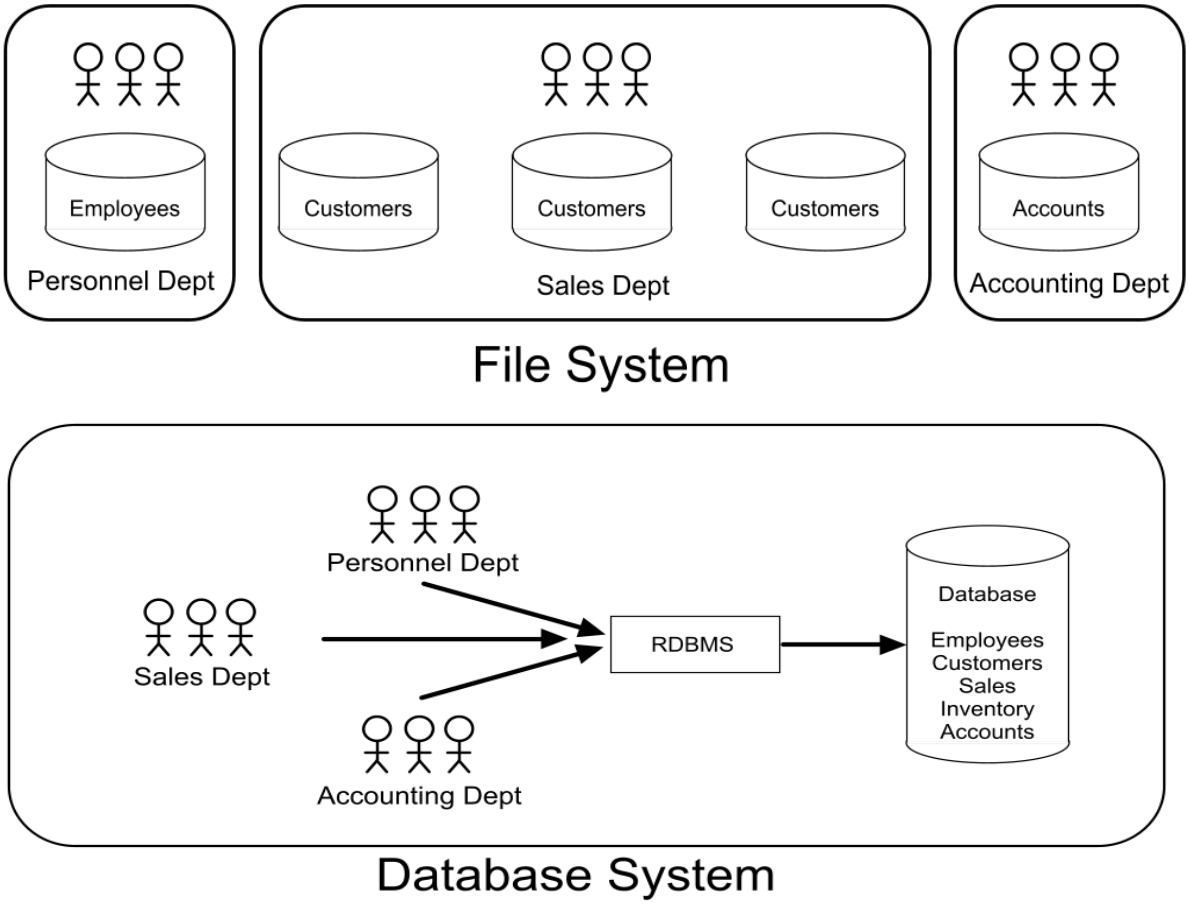
C_NAME	C_PHONE	C_ADDRESS	C_ZIP	A_NAME	A_PHONE	TP	AMT	REN
Alfred A. Ramas	615-844-2573	218 Fork Rd., Babs, TN	36123	Leah F. Hahn	615-882-1244	T1	100.00	05-Apr-2010
Leona K. Dunne	713-894-1238	Box 12A, Fox, KY	25246	Alex B. Alby	713-228-1249	T1	250.00	16-Jun-2010
Kathy W. Smith	615-894-2285	125 Oak Ln, Babs, TN	36123	Leah F. Hahn	615-882-2144	S2	150.00	29-Jan-2011
Paul F. Olowski	615-894-2180	217 Lee Ln., Babs, TN	36123	Leah F. Hahn	615-882-1244	S1	300.00	14-Oct-2010
Myron Orlando	615-222-1672	Box 111, New, TN	36155	Alex B. Alby	713-228-1249	T1	100.00	28-Dec-2010
Amy B. O'Brian	713-442-3381	387 Troll Dr., Fox, KY	25246	John T. Okon	615-123-5589	T2	850.00	22-Sep-2010
James G. Brown	615-297-1228	21 Tye Rd., Nash, TN	37118	Leah F. Hahn	615-882-1244	S1	120.00	25-Mar-2011
George Williams	615-290-2556	155 Maple, Nash, TN	37119	John T. Okon	615-123-5589	S1	250.00	17-Jul-2010
Anne G. Farriss	713-382-7185	2119 Elm, Crew, KY	25432	Alex B. Alby	713-228-1249	T2	100.00	03-Dec-2010
Olette K. Smith	615-297-3809	2782 Main, Nash, TN	37118	John T. Okon	615-123-5589	S2	500.00	14-Mar-2011

All these data inconsistencies are not desirable and not advised.

Database Systems

Unlike the file system, with its many separate and unrelated files, the database system consists of logically related data stored in a single logical data repository. The database's DBMS provides numerous advantages over file system management by making it possible to eliminate most of the file system's data inconsistency, data anomaly, data dependence, and structural dependence problems.

The figure below shows the differences between a file system and a database system:



The current generation of DBMS software stores not only the data structures, but also the relationships between those structures and the access paths to those structures – all in a central location. The DBMS may be referred to as the database system's heart. Just as it takes more than a heart to make a human being function, it takes more than a DBMS to make a database system function.

1.7 The Database System Environment

Database system refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts:

- Hardware – refers to all of the system's physical devices.
- Software – for a database system to function fully, three types of software are needed:
- Operating system software,
- DBMS software, and
- Application programs and utilities.
- People – the users of the database system; system administrators; database administrators; database designers; system analysts and programmers; and end users.
- Procedures – instructions and rules that govern the design and use of the database system.
- Data – the collection of facts stored in the database.

A database system adds a new dimension to an organization's management structure, which depends on the organization's size, its functions, and its corporate culture. Database systems can be created and managed at different levels of complexity and with varying adherence to precise standards.

DBMS Functions

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database.

- Data dictionary management: The DBMS stores definitions of the data elements and their relationships (metadata) in a **data dictionary**. All programs that access the data in the database work through the DBMS. The DBMS uses the data dictionary to look up the required data component structures and relationships, relieving you from having to code such complex relationships in each program.

- Data storage management: The DBMS create and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A modern DBMS provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. **Performance tuning** relates to the activities that make the database perform more efficiently in terms of storage and access speed.

- Data transformation and presentation: The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of making a distinction between the logical data format and the physical data format.

- Security management: The DBMS creates a security system that enforces user entity and data privacy. Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform.

- Multiuser access control: To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of the database.

- Backup and recovery management: The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure.

- Data integrity management: The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data relationships stored in the data dictionary are used to enforce data integrity.

- Database access languages and application programming interfaces: The DBMS provides data access through a query language. A **query language** is non-procedural language – one that lets the user specify what must be done without having to specify how it is to be done. This is where SQL (Structured Query Language) comes into use.

- Database communication interfaces: End users can generate answers to queries by filling in screen forms through their preferred Web browser; The DBMS can automatically publish predefined reports on a Website; The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

Managing the Database System: A Shift in Focus

The database system makes it possible to tackle far more sophisticated uses of the data resources, as long as the database is designed to make use of that available power. The kinds of data structures created within the database and the extent of the relationships among them play a powerful role in determining the effectiveness of the database system.

Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages:

- *Increased costs*: the cost of maintaining the hardware, software, and personnel required to operate and manage a database system.

- *Management complexity*: database systems interface with many different technologies and have a significant impact on a company's resources and culture. Changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives.
- *Maintaining currency*: to keep system current, perform frequent updates and apply the latest patches and security measures to all components.
- *Vendor dependence*: given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors.
- *Frequent upgrade/replacement cycles*: DBMS vendors frequently upgrade their products by adding new functionality. Upgrades cost money, as well as to train database users and administrators to properly use and manage the new features.

1.8 Summary

- Databases are collections of data that allow computer-based systems to store, manage and retrieve data very quickly
- Data are raw facts that have not yet been processed to reveal their meaning
- Information is the result of processing raw data to reveal its meaning
- A database is a shared, integrated computer storage structure that stores:
  1. Raw data
  2. Metadata
- The Database Management System (DBMS) serves as the intermediary between the user and the database. It performs the following functions:
  - Receives all application requests and translates them into the complex operations required to fulfil those requests
  - Hides much of the database's internal complexity from the application programs and users
- There are numerous types of databases, such as:
  - Single-user databases
- Multiuser databases
- Workgroup databases
- Enterprise databases
- Centralized databases
  - Distributed databases
  - Operational databases
  - Data warehouses
- A well-designed database facilitates data management and generates accurate and valuable information
- Prior to databases, information was stored in file systems
- The file system has some disadvantages, such as:

- It requires extensive programming
- There are no ad-hoc query capabilities
- System administration can be complex and difficult
- It is difficult to make changes to existing structures
- Security is usually inadequate
- A database system is composed of the five major parts:
  - Hardware
  - Software
  - People
  - Procedures
  - Data
- A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database
- Data dictionary management
- Data storage management
- Data transformation and presentation
- Security management
- Multiuser access control
- Backup and recovery management
- Data integrity management
- Database access languages and application programming interfaces
- Database communication interfaces
- Database systems carry some significant disadvantages:
  - Increased costs
  - Management complexity
  - Maintaining currency
  - Vendor dependence
  - Frequent upgrade/replacement cycles

Revision Questions

1. Define each of the following terms:
   a. data
   b. field
   c. record
   d. file
2. What is data redundancy, and which characteristics of the file system can lead to it?
3. What is the role of a DBMS, and what are its advantages?
4. What are the disadvantages of a database system?
5. What is data independence, and why is it lacking in file systems?
6. What common problems does a collection of spreadsheets created by end users share with the typical file system?

1.9 Answers to Activities

Definitions are:

a. *Data:* These are raw facts that have little meaning unless they have been organized in some logical manner.
b. *Field:* A character or a group of characters that has a specific meaning, used to define and store data.
c. *Record:* This is a logically connected set of one or more fields that describes a person, place, or thing.
d. *File:* This is a collection of related records.

2. Data redundancy exists when the same data are stored unnecessarily at different places. The fact that files in a file system cannot link to each other and share data between them necessarily means that data has to be stored in multiple places.

3. A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database.
   a. Data dictionary management: The DBMS stores definitions of the data elements and their relationships (metadata) in a **data dictionary**. All programs that access the data in the database work through the DBMS. The DBMS uses the data dictionary to look up the required data component structures and relationships, relieving you from having to code such complex relationships in each program.

   b. Data storage management: The DBMS create and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A

modern DBMS provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. **Performance tuning** relates to the activities that make the database perform more efficiently in terms of storage and access speed.

c. Data transformation and presentation: The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of making a distinction between the logical data format and the physical data format.

d. Security management: The DBMS creates a security system that enforces user entity and data privacy. Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform.

e. Multiuser access control: To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of the database.

f. Backup and recovery management: The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure.

g. Data integrity management: The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data relationships stored in the data dictionary are used to enforce data integrity.

h. Database access languages and application programming interfaces: The DBMS provides data access through a query language. A **query language** is non-procedural language – one that lets the user specify what must be done without having to specify how it is to be done. This is where SQL (Structured Query Language) comes into use.

i. Database communication interfaces: End users can generate answers to queries by filling in screen forms through their preferred Web browser; The DBMS can automatically publish predefined reports on a Website; The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

4. Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages:

a. *Increased costs*: the cost of maintaining the hardware, software, and personnel required to operate and manage a database system.
b. *Management complexity*: database systems interface with many different technologies and have a significant impact on a company's resources and culture. Changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives.
c. *Maintaining currency*: to keep system current, perform frequent updates and apply the latest patches and security measures to all components.
d. *Vendor dependence*: given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors.
e. *Frequent upgrade/replacement cycles*: DBMS vendors frequently upgrade their products by adding new functionality. Upgrades cost money, as well as to train database users and administrators to properly use and manage the new features.

5. Data independence exists when it is possible to make changes in the data storage characteristics without affecting the application program's ability to access the data. File systems are structured in such a way that the external programs which access the data in the files are entirely dependent on the files.

6. While a spreadsheet allows for the creation of multiple tables, it does not support even the most basic database functionality such as support for self-documentation through metadata, enforcement of data types or domains to ensure consistency of data within a column, defined relationships among tables, or constraints to ensure consistency of data across related tables. Most users lack the necessary training to recognize the limitations of spreadsheets for these types of tasks. As a result, they attempt to use spreadsheets like databases, which leads to problems similar to normal file systems, such as:
    a. Data redundancy
    b. Structural dependence
    c. Data dependence

Unit 2:

Data Models

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
2.1 Data Models	• Understand the concept of data modelling and know why data models are important
2.2 Data Modelling and Data Models	• Understand the basic data-modelling building blocks
2.3 The Importance of Data Models	• Explain what business rules are and how they influence database design
2.4 The Basic Building Blocks of a Data Model	• Understand how the major data models evolved
2.5 Business Rule	• Explain how data models can be classified by level of abstraction
2.6 Summary	• Summarise topic areas covered in unit

Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

2.1 Data Models

Database modelling connects real-world objects with the database stored in the computer. A problem with database design is that designers, programmers and end users see data in different ways. Different views of data can result in quite different database designs, none of which may serve purpose it was intended for fully. To avoid this, designers must get an accurate description of the nature of the data as well as the many uses of that data within the organization. To achieve this, they use **Data Models**.

2.2 Data Modelling and Data Models

Database design focuses on how the database structure will be used to store and manage end-user data. Data modelling is the first step in designing a database. It is the process of creating a specific top-level graphical representation of a specified problem domain. A problem domain is a clearly defined area within the real-world environment which well-defined scope and boundaries that must be tackled systematically. A data model is a simple, graphical representation of more complex real-world structures.

A model is an abstraction of a more complex real-world object or event. It helps you to understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations and other constructs. It supports a specific problem domain.

Data modelling is the process of creating a data model. It is an iterative, progressive process. It is essential to firstly understand the problem domain. As your understanding of the problem domain increases, the level of detail of the problem also increases. The final data model is a "blueprint" that contains all the instructions to build a database that meets all end-user requirements. The blueprint contains text descriptions and clear, useful diagrams that represent the main data elements.

A complete, implementation-ready data model should contain the following components:

1. A description of the data structure that will store the end-user data.
2. A set of enforceable rules to guarantee the integrity of the data.
3. A data manipulation methodology to support the real-world data transformation.

2.3 The Importance of Data Models

Prescribed and Recommended Textbooks/Readings

Read section 2.1 of Rob et al to get a firm understanding of the importance of data models.

Data models make communication among the designer, applications programmer and end user simpler. A well-developed data model can also improve understanding of the organization for which the database is developed. Data models are essentially a communication tool.

Data are the most basic information units used by a system. Applications are created to manage data and help transform the data into information. However, data is viewed in different ways by different people. Also, different managers of the same company view data differently, depending on their usage of the data. On the other hand, applications programmers are more concerned with data location, formatting and specific reporting requirements. They study company documents from many different sources and represent them as suitable interfaces, reports and query screens. An effective data model unifies all of these views into one.

It is important to remember that a data model is a mere abstraction of a database, very much like the architectural plans of a house; it is not the database itself, hence the required data cannot be obtained from the data model. However, a good database cannot be created without creating a suitable data model first.

2.4 The Basic Building Blocks of a Data Model

The basic building blocks of all data models are:

1. Entities
2. Attributes
3. Relationships
4. Constraints

An **entity** is anything about which data can be collected, such as a person, a place, a thing or an event. It represents a specific type of object in the real world. Each entity occurrence is unique and distinct.
An **attribute** is characteristic of an entity. Attributes are the same as fields in file systems.
A **relationship** describes a connection among entities. Data models use three types of relationships:

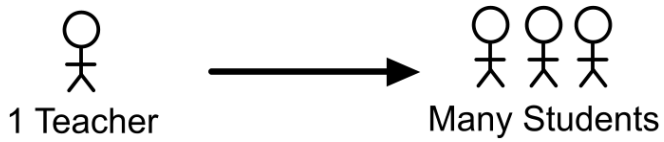
1. One-to-many
2. Many-to-many
3. One-to-one

Relationships are bidirectional, i.e. a relationship can be identified in both directions.
A **constraint** is a limit that is placed on the data. They are important because they help to ensure data integrity (the correctness of the data). Constraints are expressed in the form of rules.
To properly identify entities, attributes, relationships and constraints the first step is to identify the business rules for the problem domain you are modelling.

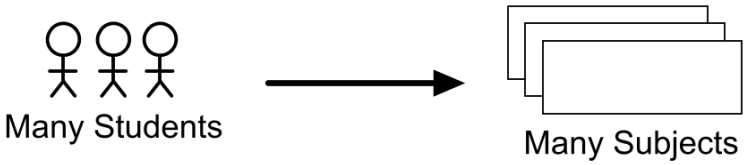
To understand the above concepts, consider a database which is being developed for a school management system. In such a system a typical **entity** would be a student, a teacher, a subject and a classroom, among others.

Each entity will have its own **attributes**, such as a student's name, surname, student number, grade, etc.
The entities in the system do not exist in isolation, and will necessarily interact with each other. As such, they will be linked to each other in various ways. For example:
"A teacher teaches a group of students a subject in a classroom"

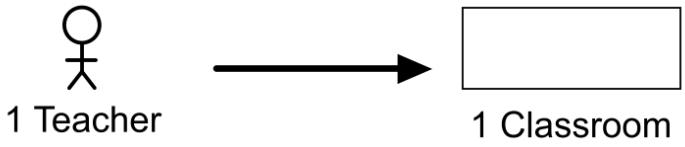
In the scenario, one teacher teaches many students, so we say that there is a **one-to-many** relationship between the teacher entity and the student entities.



In a similar way, there are numerous students, and each will take various subjects. This can be modelled via a **many-to-many** relationship between the student entities and the subject entities.



Lastly, if each teacher is allocated a classroom, and each classroom is used by that teacher only, we say that there is a **one-to-one** relationship between the teacher entity and the classroom entity.



As far as the constraints are concerned, there may be limits on the number of subjects that a student may take; for example, a student must take at least one subject, and at most six. Also, the number of subjects simply cannot be less than zero.

2.5 Business Rules

Data only has meaning when it properly reflects defined business rules. A business rule is a short, accurate and clear description of a policy, procedure or principle within a specific organization. Business rules apply to any organization, large or small, that stores and uses data to generate information. These organizations may include a business, a government unit, a religious group or a research laboratory.

Business rules are obtained from a detailed description of an organization's operations. They help to create and execute actions within the organization's environment. They must be written and updated to show any change in the organization's operational environment. Well written business rules are used to identify entities, attributes, relationships and constraints.

To be effective, business rules must be easy to understand and distributed to as many involved people as possible to make sure that every person in the organization shares a common understanding of the rules. Business rules describe the main and distinguishing characteristics of the data as viewed by the company.

Examples of business rules are:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training schedule cannot be scheduled for fewer than 10 employees or more than 30 employees.

Discovering Business Rules

The main sources of business rules are:

- People involved in the company such as, such as company managers, policymakers, department managers
- Written documentation such as a company's procedures, standards and operations manuals.

There are other sources of business rules, such as direct interviews with end users. Direct interviews are certainly a faster and more direct source of business rules, but because there are different views among each person, end users may be a less reliable source for specifying business rules. For example, the school's treasurer will only be interested in the financial aspects of the school, and will not at all be concerned with other things like student marks. So if she were to be interviewed, she will naturally give a rather one-dimensional description of the business rules, which will revolve around the financial aspects only.

Nonetheless, if end users are interviewed, and it is found that their descriptions are incomplete or contradictory, then it will be database designer's job to resolve the differences and confirm the results to ensure the business rules correct and clear.

Identifying and documenting business rules, is important to database design for the following reasons:

- They standardize the company's view of data.
- They are a communication tool between users and designers.
- They allow the designer to understand the nature, role and scope of the data.
- They allow the designer to understand business processes.

- They allow the designer to develop suitable relationship participation rules and constraints and also to create an accurate data model.

Translating Business Rules into Data Model Components

A noun in a business rule will translate into an entity in the model. A verb (active or passive) connecting nouns will translate into a relationship among entities. To properly identify the type of relationship, always remember that relationships are bidirectional, i.e. they go both ways.

Naming Conventions

When identifying entities, attributes, relationships and constraints, you also have to name the object in a way that makes it unique and clearly noticeable from other objects in the problem domain. Therefore, it is important to name the objects carefully. Entity names should describe the objects in the business environment and use words that are familiar to the users. Attribute names should also describe the data represented by the attribute. Using a proper naming convention improves the data model's ability to simplify communication among the designer, application programmer and end user. It also makes the data model self-documenting.

The Evolution of Data Models

As computers became more widely used, and the amount of information they stored grew, it became necessary to find better, more efficient ways of storing, processing and retrieving data. This led to several many different database models being developed over time, each offering certain improvements on the previous models.

The following table summarizes the evolution of these models through the decades:

Generation	Year	Model
First	1960-1970s	File system
Second	1970s	Hierarchical and network
Third	Mid 1970s to present	Relational
Fourth	Mid 1980s to present	Object-oriented

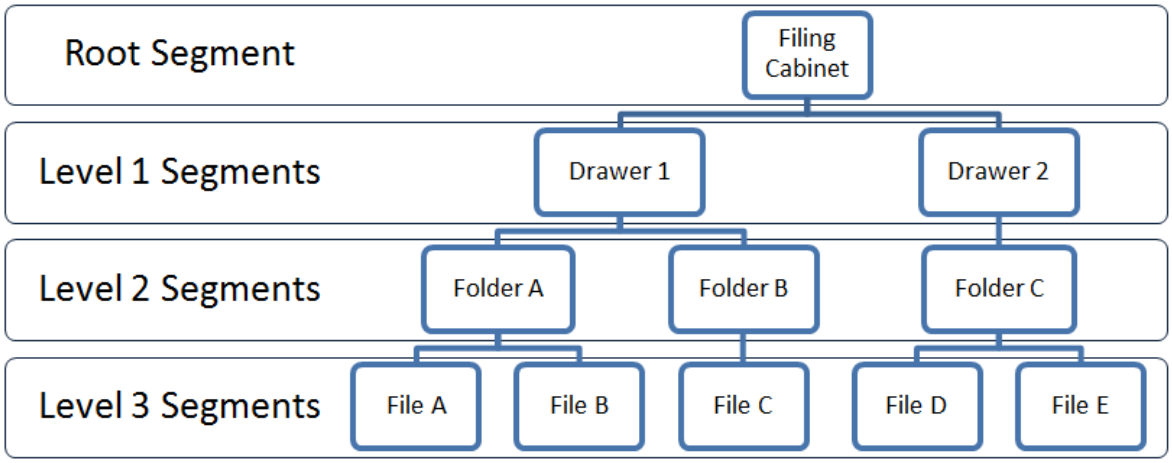
Prescribed and Recommended Textbooks/Readings

Table 2.1 of the prescribed book contains a detailed explanation of the evolution of data models throughout the decades since computers began to be used for commercial purposes. It is important to understand this evolution process in order to fully appreciate the nature of modern database systems.

Study section 2.4 of the prescribed book, paying careful attention to Table 2.1.

The Hierarchical Model

The **hierarchical model** was initially developed to manage large amounts of data for complex manufacturing projects. As the term hierarchy suggests, its logical structure is represented by an upside-down tree. The hierarchical structure contains levels, or segments. A segment is the same as a file's system's record type. A higher layer is the parent of the segment directly below it. The lower layer is called the child. The hierarchical model represents a set of one-to-many (1:M) relationships between a parent and its children segments. Each parent can have many children, but each child has only one parent. The following diagram shows how a filing cabinet may be arranged in a hierarchical form:



In the diagram, the filing cabinet itself forms the root segment, while the drawers form the 'children' of the root segment. The folders within each drawer are its children, and the folders themselves are parents to the files which they contain.

As you can see, this model presents a significant improvement over the traditional file system. As a result, it became very popular in mainframe computers in the 1970s. In fact, even though the hierarchical model has been replaced with more modern models, many of the features of modern data models are derived from the hierarchical model.

The hierarchical model, despite its benefits, did have many drawbacks:

- It was complex to implement
- Difficult to manage
- Lacked structural independence
- Many common data do not conform to the one-to-many form

As a result of the drawbacks of the hierarchical model, database professionals developed alternste models such as the network model.

The Network Model

The **network model** represents complex data relationships more effectively than the hierarchical model. It also improves database performance and imposes a database standard. It represents a one-to-many (1:M) relationship. However, unlike the hierarchical model, the network model allows a record to have more than one parent.

The specifications for the network model contained specifications for three crucial database components:

- The **schema** is the conceptual organization of the database from the database administrator's view.
- The **subschema** defines the part of the database that is seen by the application programs that produce the information from the data within the database.
- A **data management language (DML)** defines the environment in which data can be managed. It also works with the data in the database.
- A schema **data definition language (DDL)** allows the database administrator to define the schema components.

As information needs grew the network model proved to be just as cumbersome as the hierarchical model. This led to both models being replaced by the relational database model.

The Relational Model

The relational database model caused a database revolution due to its simplicity and effectiveness. Although it was first used on mainframe computers, today it is widely used on personal computers and even mobile devices.

One of the most powerful aspects of the relational databases is that they are implemented by, and almost entirely managed by a **relational database management system (RDBMS)**. The RDMS acts as the "middle man" in a relational database, and performs all the complex internal functions of the database such as creating the database and its components, physical storage of the data in the database, and sorting and searching the database. As a result, the user can manipulate and query the data in an intuitive and logical way. The complexities of the database functions are all hidden from the user and the developer.

The name "relational model" comes from the fact that in a relational database the data are stored in tables, and tables with common data are "related" to each other by sharing a common attribute (a value in a column). A table is also known as a relation.

Consider the example below which demonstrates tables in a relational database:

Table name: SALESPERSON

SALESPERSON_CODE	SALESPERSON_SURNAME	SALESPERSON_FIRSTNAME
1001	Smith	Frank
1002	Lee	James
1003	Evans	Adam

Table name: CUSTOMER

CUS_CODE	CUS_SURNAME	CUS_FIRSTNAME	CUS_AREACODE	CUS_TEL	SALESPERSON_CODE
10010	Brown	Mary	031	5551234	1002
10011	Orlando	Peter	011	6667890	1003
10012	Fariss	Akmal	012	3456789	1003
10013	Dunne	Roger	011	5544321	1001

The tables above represent different data of a business, but they share a common link: the SALESPERSON_CODE. The first column in the SALESPERSON table and the last column in the CUSTOMER table contain the SALESPERSON_CODE field, which links the two tables together. What this actually means is that each customer is allocated a salesperson with whom he deals. The id of the salesperson is indicated in the SALESPERSON_CODE column of the CUSTOMER table. So, for example, the customer Mary Brown deals with the salesperson whose SALESPERSON_CODE is 1002. Going through the SALESPERSON table, we can see that the salesperson being referred to is James Lee; but not just that - we can retrieve any information about that particular salesperson.

Clearly this is a far more effective system than the file system, which would necessitate that all salesperson information, including name, surname, etc. should be replicated in the customer file as well.

Like a file, a relational table stores a collection of related entities. But a table is very different to a file in some crucial ways: a table produces complete data and has structural independence. In other words, the table is a logical representation of the data, and not how it is actually stored in the system. The physical structure of the data is handled by the RDBMS, and is of no interest to the user.

The relational data model also has a powerful and flexible query language called **Structured Query Language (SQL)**. SQL (officially pronounced S-Q-L, but also pronounced “sequel”) allows the user to specify what must be

done without specifying how it must be done. The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data.

A SQL-based relational database application has three parts:

1. *The end-user interface*: allows the user to interact with the data. It does this by auto-generating SQL code.
2. *A collection of tables stored in the database*: In a relational database, all data are stored in tables. The tables display the data to the end user in a way that is easy to understand. Each table is independent. Rows in different tables are related by common values in common attributes.
3. *SQL engine*: executes all queries or data requests. It is part of the DBMS software.

The Entity Relationship Model

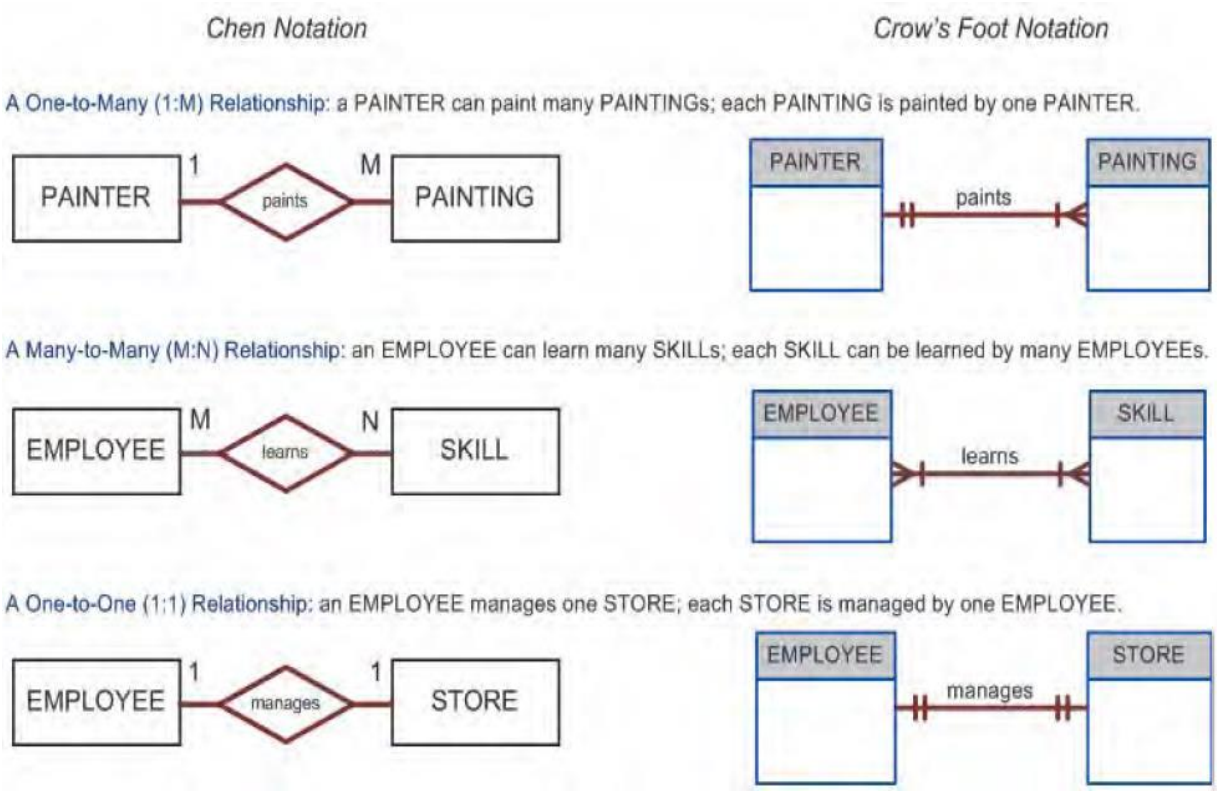
The success of the relational database model made it possible to conveniently store ever-increasing amounts of data digitally, and databases became very large and complex. With this complexity came the need to develop effective methods of carefully planning and designing databases. One of the most effective ways to do this was via entity relationship modelling.

The **entity relationship (ER)** model **(ERM)** is a graphical representation of entities and their relationships. An **entity relationship diagram (ERD)** represents model database components.

The ER model is based on the following components:

- *Entity*: is represented in the ERD by a rectangle called an entity box. Each row in the relational table is known as an **entity instance** or **entity occurrence** in the ER model. Each entity is described by a set of attributes. These attributes describe the characteristics of each entity.
- *Relationships*: describe associations among data. Most relationships describe associations between two entities.

The following diagram shows the different types of relationships using two ER notations: the original Chen notation and the current Crow's Foot notation (you will learn more about these notations in chapter 4 of this module):



The Object-Oriented (OO) Model

Think Point

Understanding Object-oriented programming

In order to understand the concept of object-oriented database models, it is essential to thoroughly understand the concept of object-oriented programming. The concept of OO programming was dealt with in detail in Informatics 1B. Study the relevant sections in that module in order to refresh your understanding of the topic.

There are also numerous web resources to get a better understanding of OO concepts. Look these up via Google.

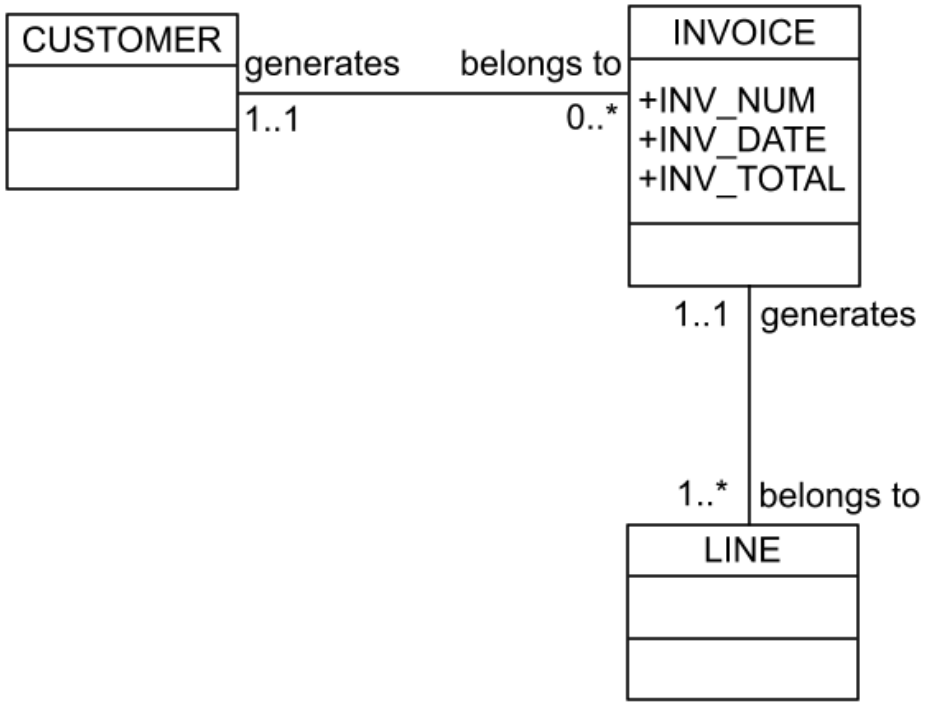
The ever-increasing complexity of real-world problems necessitated the development of a system of storing data in a way which resembled the real world. As a result, the object-oriented paradigm in software development and databases was developed. The **object-oriented data model (OODM)** represents the real world more closely than the relational model. Data and their relationships are contained in a structure called an **object**. In an OODM, an object is similar to the relational model's entity in that it is described by its content. However, an object is different from an entity because it includes information about an object as well as information about its relationships with other objects. Therefore, the facts within the object have greater meaning.

An object-oriented data model is based on the following components:

- *Object*: An object is an abstraction of a real-world entity. It represents one occurrence of an entity.
- *Attributes*: An attribute describes the properties of an object.
- *Class*: A **class** is a collection of similar objects with shared structure (attributes) and behaviour (methods).
- *Method:* A **method** represents a real-world action. It defines an object's behaviour.
- *Class hierarchy*: A **class hierarchy** is an upside-down tree in which each class has only one parent.
- *Inheritance*: **Inheritance** allows an object within a class hierarchy to inherit the attributes and methods of the classes above it.

Object-oriented data models are depicted using Unified Modelling Language (UML) class diagrams.

Unified Modelling Language (UML) is a language based on OO concepts. It describes a set of diagrams and symbols that are used to graphically model a system. UML **class diagrams** represent data and their relationships within the larger UML object-oriented system's modelling language.
The diagram below shows a UML diagram of a very basic invoicing system:



Note the following in the diagram:

- The UML class diagram uses three separate object classes (CUSTOMER, INVOICE, and LINE) and two relationships to represent this simple invoicing problem. Note that the relationship connectivities are represented by the 1..1, 0..*, and 1..* symbols and that the relationships are named in both ends to represent the different "roles" that the objects play in the relationship.
- The ER model also uses three separate entities and two relationships to represent this simple invoice problem.

Google's scary powerful Spanner database gets us one step closer to Skynet

Google's Spanner is a single database that runs across hundreds of data centres throughout the world. It's so smart that it rapidly shifts resources during outages without human intervention and keeps everything perfectly in sync using GPS and atomic clocks.

While Spanner was introduced a few months back, a detailed profile from Wired today explores more about how Spanner works and why it's revolutionary. Google has spent four and half years on Spanner, and it's hard to imagine where the project will be in a few more years.
Here are four big points that show what makes Spanner so cool:

1. Spanner replicates Google's data across multiple data centres, and Google services can pull from these replicas as needed.

2. Because Spanner can replicate data so easily, it makes Google infrastructure more resistant to "network delays, data-centre outages, and other software and hardware snafus."

3. To get the best timing accuracy possible, Google installed GPS antennas on top of its many data centres and connected them to the millions of machines inside those centres. In case the GPS fails to keep accurate time (during an extreme weather event, for example), there are atomic clocks there too to keep time perfect.

4. Google's ad network (which generates the vast majority of the company's cash) benefits greatly from Spanner. These auctions need to have incredibly precise time, especially with some auctions decided by milliseconds.

Spanner isn't quite Skynet — the self-aware artificial intelligence system in the Terminator movies — but it does show how far we've come at building connected systems and databases.

"When there are outages, things just sort of flip — client machines access other servers in the system," Google software engineer Andrew Fikes told Wired. "It's a much easier service story. … The system responds — and not a human."

Source http://venturebeat.com/2012/11/26/google-spanner-skynet/#MA66zjOproCpyQrf.99

Prescribed and Recommended Textbooks/Readings

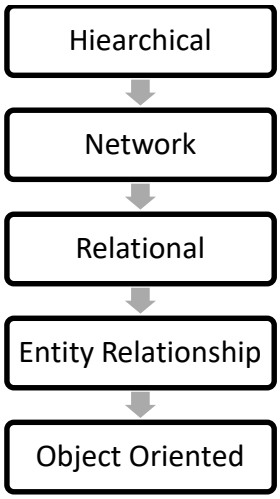
Database models and the internet

Read section2.4.7 on page 52 of Rob et al to gain an understanding into the impact of the internet on databases.

Data Models: A Summary

Common characteristics of data models that make them widely-accepted:

- A data model must show some degree of conceptual simplicity compromising the semantic completeness of the database. The data model must not be more difficult to interpret that the real world.
- A data model must represent the real world as closely as possible.
- Representation of the real-world transformations (behaviours) must match the consistency and integrity characteristics of any data model.

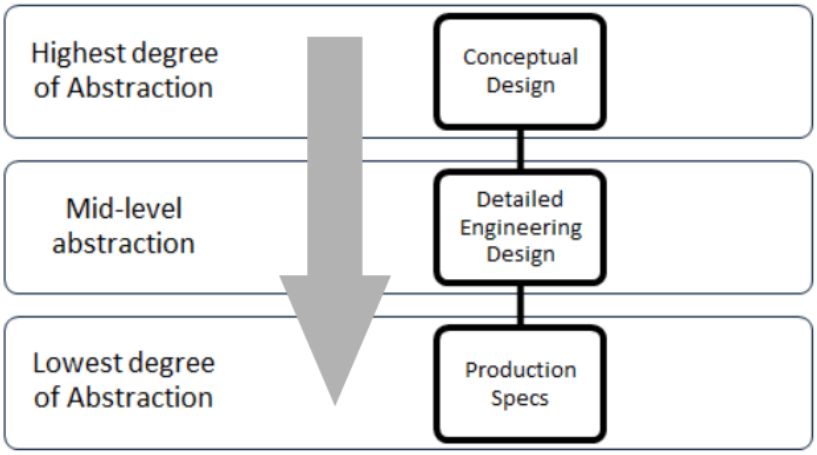


Each new data model addresses the problems of the older data models. The network model replaced the hierarchical because it makes it easier to represent complex (many-to-many) relationships. The relational model is better than the hierarchical and network models because it has simpler data representation, improved data independence and easy-to-use query language. As a result, it is the preferred model for business applications. The OO data model supports complex data within a rich semantic data framework. The ERDM added many OO features to the relational model and allowed it to maintain its strong market share within the business environment.

Degrees of Data Abstraction

In order to simplify designing of a database, a database designer may model the data based on degrees of data abstraction. Abstraction starts at the highest degree, and progresses to lower and lower degrees. To understand the concept of abstraction, think about the process of manufacturing a car; the process always begins with a

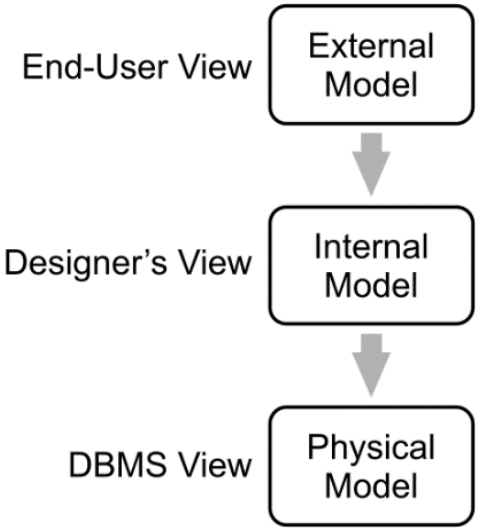
concept drawing of the car, then moves to detailed engineering designs and finally to extremely detailed production specifications of each part of the car, which is then built and assembled according to those specs. This is illustrated in the diagram below:



A database is ideally designed in the exact same way – from top to bottom. First the designer will look at a high level concept of the database, and will then add more and more details as the design gets closer to being implemented. This process is known as data abstraction.

An abstraction is very useful in integrating multiple (and sometimes conflicting) views of data as seen at different levels of an organization.

As far as data abstraction goes, there are essentially three degrees of abstraction: external, conceptual and internal. These are shown in the diagram below:



The External Model

The **external model** is the end users' view of the data environment. End users are people who use the application programs to handle the data and generate information. They usually work in an environment where an application has a specific business unit focus. Each business unit has specific constraints and requirements. Also, each unit uses a subset of the total data in the organization; therefore they view their data subsets as separate from or external to other units within the organization.

ER diagrams are used to represent the external views. An **external schema** is a specific representation of an external view.

Important advantages of external views:

- Makes it easy to identify specific data required to support each unit's business operations.
- Makes the designer's job easy by providing feedback about the model's adequacy. The model can be checked to ensure that it supports all processes as defined by their external models, as well as operational requirements and constraints.
- Helps to ensure security constraints in the database design.
- Simplifies application program development.

The Conceptual Model

The **conceptual model** represents a general view of the whole database as the organization sees it. It combines all the external views into a single view. The conceptual model is also known as a **conceptual schema**, and is the basis for identification and high-level description of the main data objects. It avoids any database model-specific details. In other words, it is concerned with what the data needs of the organization are, not how they will be implemented.

The conceptual model is graphically represented by an Entity-Relationship Diagram (ERD).

Advantages of the conceptual model:

1. It makes the data environment easy to understand.
2. It is software-independent
3. It is hardware-independent

Software independence refers to the fact that the model does not depend on the DBMS software used to implement the model. **Hardware independence** refers to the fact that the model does not depend on the hardware used in the implementation of the model.

The Internal Model

The **internal model** maps the conceptual model to the DBMS. It represents the database as the DBMS understands it. An **internal schema** is a specific representation of an internal model. It uses the database constructs supported by the chosen database.

The internal model is software-dependent because it depends on specific database software. **Logical independence** refers to changing the internal model without affecting the conceptual model. The internal model is hardware-independent.

The Physical Model

The **physical model** operates at the lowest level of abstraction. It describes the way data are saved on storage media such as disks or tapes. It is both software- and hardware-dependent. Database designers working at this level must have a detailed knowledge of the hardware and software used to implement the database design.

2.6 Summary

- Database designers use data models get an accurate description of the nature of the data as well as the many uses of that data within the organization.
- The basic building blocks of all data models are:
  1. Entities
  2. Attributes
  3. Relationships
  4. Constraints
- Identifying and documenting business rules, is important to database design for the following reasons:
- They standardize the company's view of data.
- They are a communication tool between users and designers.
- They allow the designer to understand the nature, role and scope of the data.
- They allow the designer to understand business processes.
- They allow the designer to develop suitable relationship participation rules and constraints and also to create an accurate data model.
- Over time, many data models were developed, such as:
- File systems
- Hierarchical models
- Network models
- Relational models
- Object-oriented models
- In order to simplify designing of a database, a database designer may model the data based on degrees of data abstraction.
- There are essentially three degrees of abstraction: external, conceptual and internal.

Revision Questions

1. Discuss the importance of data modelling
2. What is a business rule, and what is its importance in data modelling?
3. Describe the basic features of the relational data model and discuss their importance to the end user and the designer
4. Besides redundancy, what other problems are associated with the non-database approach to processing data?
5. What is an ER diagram?

2.7 Answers to Activities

1. A common problem with database design is that designers, programmers and end users see data in different ways, can result in contrasting database designs. To avoid this, designers must get an accurate description of the nature of the data as well as the many uses of that data within the organization. To achieve this, they use Data Models.

2. A business rule is a short, accurate and clear description of a policy, procedure or principle within a specific organization.

Identifying and documenting business rules, is important to database design for the following reasons:

- They standardize the company's view of data.
- They are a communication tool between users and designers.
- They allow the designer to understand the nature, role and scope of the data.
- They allow the designer to understand business processes.
- They allow the designer to develop suitable relationship participation rules and constraints and also to create an accurate data model.

3. A relational database model has the following features:

- Data is stored in a series of related tables
- Data is not accessed directly by the software application, but via a powerful query language called SQL
- It has structural independence
- It stores metadata, which is information about the data which is stored in the database

These features make relational databases far more easy to use and far more efficient than their predecessors. Developers do not need to spend time and effort in understanding how and where the data is stored – this is all handled by the DBMS. All they need to do is to instruct their applications to communicate with the DBMS, and the DBMS takes care of the underlying complexities of the database.

For the end user the major benefits are speed, efficiency, accuracy of the data and flexibility. Relational databases are much faster and efficient than older systems like file systems, and because redundant data is eliminated in relational databases, the data tends to be much more accurate.

4. The problems, other than redundancy, associated with the non-database approach to processing data include difficulties accessing related data, limited security features to protect data from access by unauthorized users, limited ability for multiple users to update the same data at the same time, and size limitations.

5. An E-R diagram represents a database in a visual way by using a rectangle for each entity, using a line to connect two entities that have a relationship, and placing a dot at the end of a line to indicate the “many” part of a one-to-many relationship.

Unit 3:

The Relational Database Model

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
3.1 A Logical View of Data	• Understand the concept Logical View
3.2 Keys	• Be knowledgeable on the types of Keys
3.3 Controlled redundancy	• Know the characteristics of a table
3.4 Referential integrity	• Understand the types of data integrity
3.5 Relational Set Operators	• Explain the relational operators
3.6 Data Redundancy Revisited	• Differentiate between a system catalogue and a data dictionary
3.7 Indexes	• Display knowledge on Codd's 12 rules
3.8 Codds Relational Database Rules	• Discuss Codds Relational Database Rules
3.9 Summary	• Summarise topic areas covered in unit

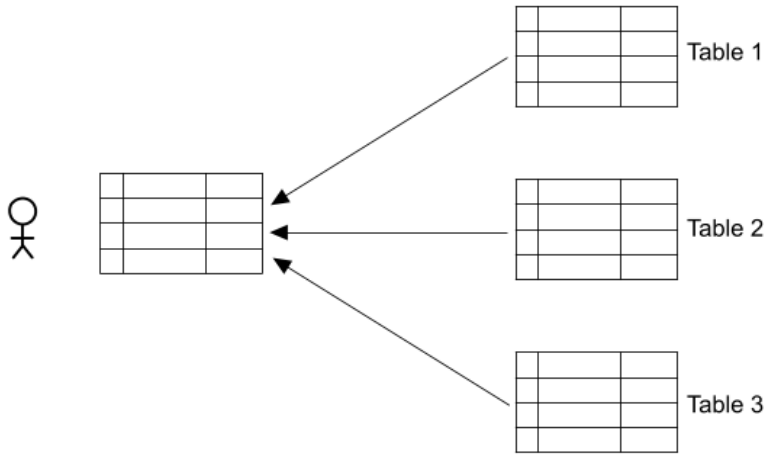
Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

3.1 A Logical View of Data

The relational model enables you to view data logically rather than physically. In other words, you view a conceptual model of how the data is stored, not how it is actually stored. Consider the following diagram:



The diagram depicts a user who is viewing data in a single table. As you can see, the data has been sourced out from multiple tables which may live on the same or different databases, and possibly in various parts of the world! None of this concerns the user, however – as far as he is concerned, the data is stored in a single table in a single file. What the user is actually seeing is a conceptual model of the data.

The logical view resembles the simple file concept of data storage with the use of tables, but unlike a file, it has the advantage of structural and data independence.

Tables and their characteristics

The logical view of the relational database is simplified by the creation of tables. Tables in a database are actually data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct which is often difficult to understand, end users find it much easier to think of it as a *table*. A table is simply perceived as a two-dimensional structure composed of rows and columns. A table contains a group of related items. These items are known as *entities*, hence a table is also referred to as *entity set*.

The characteristics of a relational table are:

- A table is perceived as a two-dimensional structure composed of rows and columns.
- Each table row (**tuple**) represents a single entity occurrence within the entity set.
- Each table column represents an attribute, and each column has a distinct name.
- Each row/column intersection represents a single data value.
- All values in a column must conform to the same data format.
- Each column has a specific range of values known as the **attribute domain**.
- The order of the rows and columns is immaterial to the DBMS.
- Each table must have an attribute or a combination of attributes that uniquely identifies each row.

To understand the above points, consider the following diagram which demonstrates a typical database table:

Table: STUDENT

Student No	Name	Surname	Telephone	Email	Registration year
3045	Sam	Lakota	0112344568	sam@samail.co.za	2008
5623	Abe	Khumalo	0315436543	abek@email.com	2010
4321	John	Smith	0326548765	js@abc.co.za	2009
4364	Mary	Watson	0125436543	mary@abc.com	2009
2097	Peter	Brown	0115673456	peterb@def.co.za	2010
4356	John	Smith	0312345678	smithj@qwety.com	2011

Note the following:

- The table is a 2-dimensional structure which is comprised of rows and columns
- Each row represents data relating to one student only
- The rows are not stored in any particular order
- Each column has a unique, descriptive name and represents one bit of data, or one single fact, about the student
- The items in each column are all of the same format – that is, if you take the Email column, it has only emails in it, while the Registration_year column has only the years listed in full format
- Each student number is unique – this is the unique attribute that identifies each student

Prescribed and Recommended Textbooks/Readings

Degree and cardinality

Degree and cardinality are two important properties of the relational model. Study section 3.1.3 of the prescribed book, ensuring that you have a good understanding of these concepts

3.2 Keys

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identifiable and establishes relationships among tables and ensures integrity of the data by preventing duplicates, for example. A **key** may consist of one or more attributes, and is used to determine other attributes. For example, the Student No in the previous table serves as a key to all the other attributes. If you know the student number of a student, you will be able to access all the other attributes such as Name, Surname, and Email etc. It is for this

reason that the key's role on a concept is known as **determination**, because the statement ***"A determines B"*** indicates that if you know the value of attribute A, you can look up (determine) the value of attribute B, and thus can be represented as A → B.

A **primary key** is an attribute, or a combination of attributes, that uniquely identifies any given row, such as the Student_No attribute in the table. As you can see, the name and surname are not unique; in the table there are two people with the name John Smith, so the name and surname attributes are not unique, and hence cannot be used as primary keys.

So, looking at the STUDENT table, if you know the value of attribute Student_No, you can determine the value of the remaining attributes.This can be represented as follows:
Student_No → Name, Surname, Telephone, Email, Registration_year

The principle of determination is very important because it is used in the definition of a central relational database concept known as **functional dependence**. In other words, an attribute B is functionally dependent on another attribute A if each value in column A determines ***one and only one*** value in column B. Using the same STUDENT table, it is appropriate to say that Telephone is functionally dependent on Student_No because the Student_No value 4321 determines the Telephone value 0326548765. However, we can also see that Student_No is *not* functionally dependent on Name because there may be many students with the same name, such as in the case of Kohn Smith. You can compare this dependency between Student_No and Surname as well.

It might take more than a single attribute to define functional dependence. Hence, a key may be composed of more than one attribute. This type of multi-attribute key is referred to as a **composite key**. Any attribute that is part of a key is known as a **key attribute**. Referring to the STUDENT table again, we can see that Name cannot be a suitable key alone. However it can be combined with other attributes to make a composite key, such as Name, Surname, Email and Telephone can be a suitable key.

Given the possible existence of a composite key, the notion of functional dependence is further refined by specifying **full functional dependence**. In other words, if the attribute B is functionally dependent on a composite key, A, but not on any subset of that composite key, the attribute B is fully functionally dependent on A.

There are other specialized keys that can be defined:

- **Superkey**: this is any key that uniquely identifies each row. A superkey can be a single attribute or a multi-attribute composite key, provided the primary key is part of that superkey. The primary key, on its own, can be a superkey. For example, each of the following can be a selected superkey:
  - Student_No
  - Student_No, Name
  - Student_No,Name,Surname

- **Candidate key**: this can be described as a superkey without unnecessary attributes, that is, it is a minimal superkey. As an example, the key Student No by itself is a candidate key. On the other hand, the combination Student No, Surname might serve as a *superkey*, but is not a candidate key because Student_No is by itself a candidate key, and the addition of Surname to it is not strictly necessary.

  In a similar way, the combination Name, Surname, Telephone, Email might also be a candidate key, provided there is no possibility of two students sharing the same combination of last name, first name, telephone number and email address.

- **Foreign key**: this refers to a value in an attribute in a table which serves as a primary key in another table, and which serves to link the two tables. This is illustrated in the following diagram, where the Manuf_Code values in the PRODUCT table are primary keys in the MANUFACTURER table. In order to find the manufacturer for product 32445, we need to firstly look at the Manuf_Code value, which in this case is 0023. Next, we need to match that value to the Manuf_Code values in the MANUFACTURER table, which will determine the name of the manufacturer, which is "Samsung Electronics"

Table: PRODUCT
Primary key: Product_Code
Foreign key: Manuf_Code

Prod_Code	Prod_Desc	Price	Stock	Manuf_Code
32445	Monitor 18 inch	799.99	25	0023
45846	Keyboard	129.99	54	0014
48572	Laptop X5	4500.00	6	0039
37350	Scanner	329.50	3	0097
68375	DVD Player	599.00	12	0023

Table: MANUFACTURER
Primary key: Manu_Code
Foreign key: none

Manuf_Code	Manuf_Name	Contact_person	Contact_Tel	Contact_Email
0097	Canon	Jay	0112345678	jay@cn.co.za
0014	Microsoft	Mike	0312345654	mike@ms.com
0023	Samsung Electronics	Linda	0129876543	linda@abc.com
0039	Dell	Sipho	0312344321	siphon@def.com

3.3 Controlled redundancy

Controlled redundancy refers to a situation where data is repeated in a table, but not unnecessarily. As an example, consider the diagram above; in column Manuf_Code of table PRODUCT, the code value 0023 appears twice. But as you can clearly see, this is because the particular manufacturer with Manuf_Code 0023 supplies more than one product.

Relational schema

A relational database can be represented by a **relational schema**. A relational schema is a textual representation of the database tables where each table is listed by its name followed by the list of its attributes in parentheses. For example, the relational schema for MANUFACTURER and PRODUCT are will be as follows:

MANUFACTURER(**Manuf_Code**, Manuf_name, Contact_person, Contact_Tel, Contact_Email)
PRODUCT(**Prod_Code**, Prod_Desc, Price, Stock, Manuf_Code)
The primary key attributes are in old and underlined.

Integrity rules

There are two types of data integrity rules that are very important in good database design: **entity integrity** and **referential integrity**.

Entity Integrity

A table exhibits **entity integrity** when each primary key value is unique. This is necessary to ensure that each row is uniquely identified by the primary key. To maintain entity integrity it is absolutely essential that there are no null values in the primary key (a null is not a zero or a space, it means "*no value*", like pressing the *ENTER* key or the *TAB* key on the keyboard to move to the next entry without making a prior entry of any kind). A null primary key effectively means that a record in a table will be extremely difficult or impossible to find.

In entity integrity, each row will have a unique identity, and foreign key values can properly reference primary key values.

3.4 Referential integrity

A table is said to have **referential integrity** if it has a foreign key which matches a primary key value in the linked table. For example, the PRODUCT table in the previous diagram contains a foreign key value 0097. This value has a matching value in the MANUFACTURER table, which indicates referential integrity.

On the other hand, if the foreign key attribute for a row in the PRODUCT table contained a value such as 0055 which does not exist in the MANUFACTURER table, then this indicates that referential integrity has been compromised, since the product refers to a manufacturer which does not exist.
Referential integrity enforces the following three rules:

1. We may not add a record to the PRODUCT table unless the Manuf_Code attribute points to a valid record in the MANUFACTURER.
2. If the primary key for a record in the MANUFACTURER table changes, all corresponding records in the PRODUCT table must be modified using a cascading update.
3. If a record in the MANUFACTURER table is deleted, all corresponding records in the PRODUCT table must be deleted using a cascading delete.

3.5 Relational Set Operators

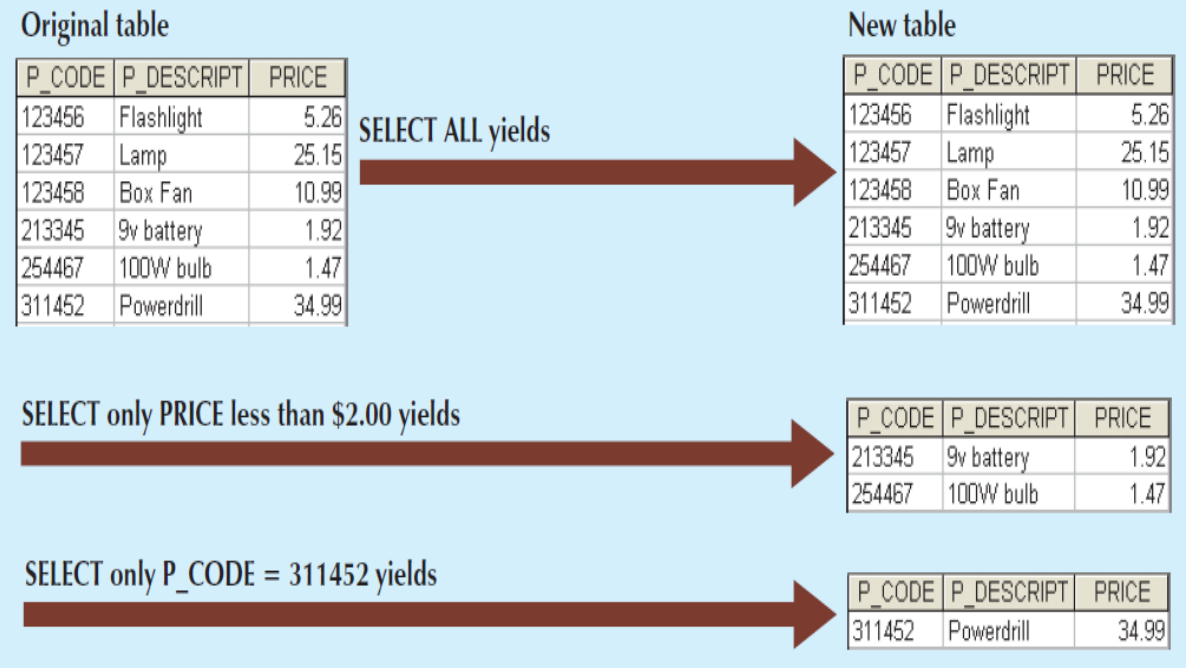
The true power of relational databases manifests itself in the fact the data in the database can be quickly and efficiently manipulated to generate useful information. The data in a table can be manipulated using a specialized set of rules known as **relational algebra**. Relational algebra, which is similar to set algebra, defines a set of eight operators which are used to manipulate relational data.

The eight relational operators are:

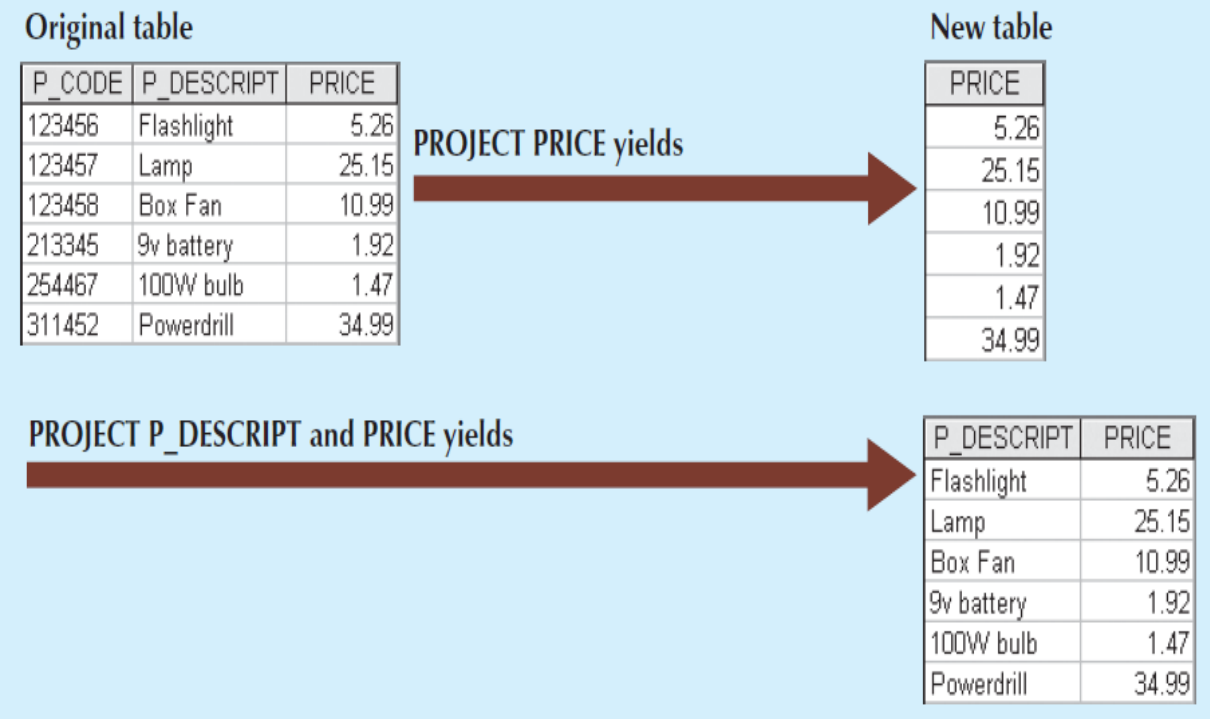
1. SELECT
2. PROJECT
3. UNION
4. INTERSECT
5. DIFFERENCE
6. PRODUCT
7. JOIN
8. DIVIDE

These operators are explained below:

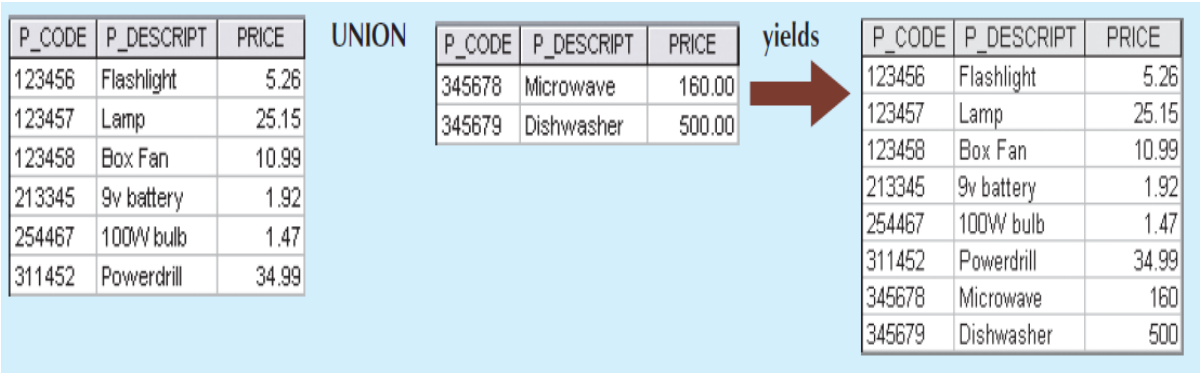
1. SELECT: returns values for all rows found in a table that satisfy a given condition. SELECT can be used to list all of the row values, or it can return only those row values that match a specified criterion. It is also known as RESTRICT:



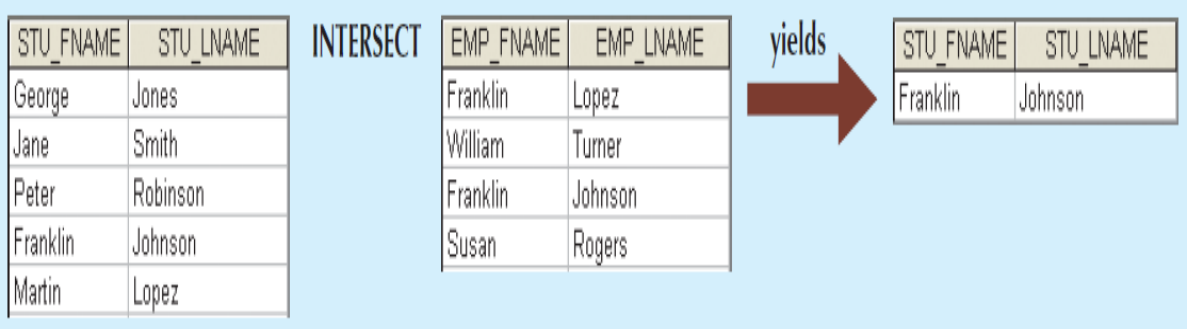
2. PROJECT: yields all values for selected attributes. In other words, PROJECT yields a vertical subset, or column, of a table:



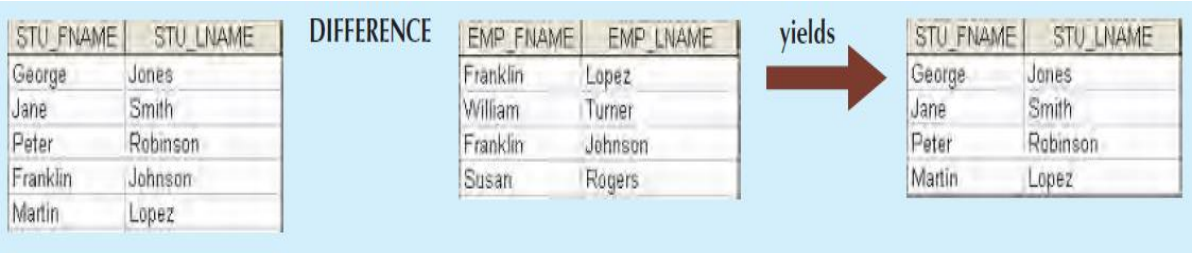
3. UNION: combines all rows from two tables, excluding duplicate rows. These tables must have the same attribute characteristics (i.e. the columns and domains must be compatible). When two or more tables share the same number of columns, and when their corresponding columns share the same (or compatible) domains, they are said to be **union-compatible**:



4. INTERSECT: yields only the rows that appear in both tables. These tables must be union-compatible to yield valid results. For example, you cannot use INTERSECT if one of the attributes is numeric and one is character-based.

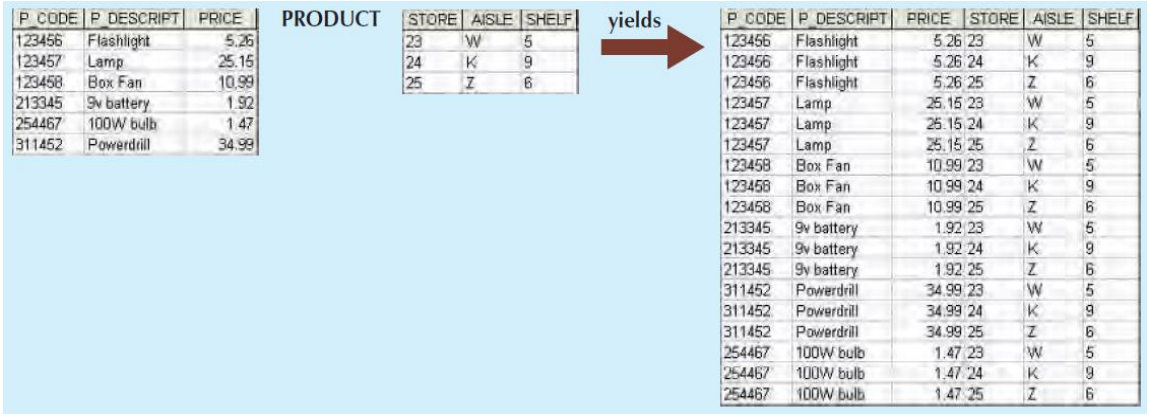


5. DIFFERENCE: yields all rows in one table that are not found in the other table; that is, it subtracts all the contents of one table from the other. These tables must also be union-compatible to yield valid results. It is important to remember that subtracting the first table from the second table is not the same as subtracting the second table from the first table:



As you can see, Franklin Johnson has been dropped off from the first table because he appears on both tables.

6. PRODUCT: yields all possible pairs of rows from two tables – also known as the **Cartesian product**. Therefore, if one table has six rows and the other table has three rows, the PRODUCT yields a list composed of 6 x 3 = 18 rows.



7. JOIN: allows information to be combined from two or more tables. The JOIN operation enables the linking of two tables with common attributes. There are four types of JOIN to consider:
    a. Natural join
    b. Equijoin
    c. Left outer join
    d. Right outer join

The following tables will be used to demonstrate the various joins:

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

a. **Natural join:** links tables by selecting only the rows with common values in their common attribute(s). A natural join is the result of a three-stage process:

1. First, a PRODUCT of the tables is created, yielding the following results:

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231	125	6152439887
1132445	Walker	32145	231	167	6153426778
1132445	Walker	32145	231	231	6152431124
1132445	Walker	32145	231	333	9041234445
1217782	Adares	32145	125	125	6152439887
1217782	Adares	32145	125	167	6153426778
1217782	Adares	32145	125	231	6152431124
1217782	Adares	32145	125	333	9041234445
1312243	Rakowski	34129	167	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1312243	Rakowski	34129	167	231	6152431124
1312243	Rakowski	34129	167	333	9041234445
1321242	Rodriguez	37134	125	125	6152439887
1321242	Rodriguez	37134	125	167	6153426778
1321242	Rodriguez	37134	125	231	6152431124
1321242	Rodriguez	37134	125	333	9041234445
1542311	Smithson	37134	421	125	6152439887
1542311	Smithson	37134	421	167	6153426778
1542311	Smithson	37134	421	231	6152431124
1542311	Smithson	37134	421	333	9041234445
1657399	Vanloo	32145	231	125	6152439887
1657399	Vanloo	32145	231	167	6153426778
1657399	Vanloo	32145	231	231	6152431124
1657399	Vanloo	32145	231	333	9041234445

2. Second, a SELECT is performed on the output of Step 1 above to yield only the rows for which the CUSTOMER.AGENT_CODE and AGENT.AGENT_CODE values are equal. The common columns are referred to as the **join columns**. The following result is shown.

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124

3. Finally, a PROJECT is performed on the results of Step 2 to yield a single copy of each attribute, thereby eliminating duplicate columns. This step yields the output shown.

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

The final outcome of a natural join yields a table that does not include unmatched pairs and provides only the copies of the matches.

2. **Equijoin**: this join links tables on the basis of an equality condition that compares specified columns of each table. The outcome of the equijoin does not eliminate duplicate columns, and the condition or criterion used to join the tables must be explicitly defined. The equijoin uses the equality comparison operator (=) in the condition. If any other comparison operator is used, the join is called a **theta join**.

The above two joins are often classified as **inner join**, a join that only returns matched records from the tables that are being joined. The next two joins are classified as **outer join**, where the matched pairs would be retained, and any unmatched values in the other table would be left null. Do not assume that an outer join is the opposite of an inner join, however, think it more of an "inner join plus", because an outer still returns all of the matched records that the inner join returns, plus it returns the unmatched records from one of the tables.

3. **Left outer join**: yields all of the rows in the CUSTOMER table, including those that do not have a matching value in the AGENT table, as shown in Figure 3.15.

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124
1542311	Smithson	37134	421	

4. **Right outer join**: yields all of the rows in the AGENT table, including those that do not have matching values in the CUSTOMER table, as shown in Figure 3.16.

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124
			333	9041234445

5. DIVIDE: uses one single-column table (e.g., column "a") as the divisor and one 2-column table (i.e., columns "a" and "b") as the dividend. The tables must have a common column (e.g., column "a"). The output of the DIVIDE operation is a single column with the values of column "a" from the dividend table rows where the value of the common column (i.e., column "a") in both tables matches.

CODE	LOC
A	5
A	9
A	4
B	5
B	3
C	6
D	7
D	8
E	8

DIVIDE

CODE
A
B

yields

LOC
5

Note that:

1. Table 1 is "divided" by Table 2 to produce Table 3. Tables 1 and 2 both contain the column CODE but do not share LOC.
2. To be included in the resulting Table 3, a value in the unshared column (LOC) must be associated (in the dividing Table 2) with every value in Table 1.
3. The only value associated with both A and B is 5.

Relational operators have a property called **closure**, which is the use of relational algebra operators on existing relations (tables) to produce new relations.

The Data Dictionary and the System Catalogue

A **data dictionary** provides a detailed description of all tables found within the user/designer-created database. The data dictionary contains at least all of the attribute names and characteristics for each table in the system. The data dictionary contains metadata – data about data.

The diagram below shows a sample data dictionary:

Table Name	Attribute Name	Contents	Type	Format	Domain	Required	PK or FK	FK Ref Table
CUSTOMER	CUS_CODE CUS_LNAME CUS_FNAME CUS_INITIAL CUS_RENEW_DATE AGENT_CODE	Customer code Surname Firstname Initial Insurance renewal date Agent code	CHAR(5) VARCHAR(20) VARCHAR(20) CHAR(1) DATE CHAR(3)	99999 Xxxxx Xxxxx X dd-mm-yy 999	10000-99999	Y Y Y	PK FK	 AGENT

Like the data dictionary, the **system catalogue** also contains metadata and can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. The terms *system catalogue* and *data dictionary* are often used interchangeably because the system catalogue contains all required data dictionary information.

The system catalogue automatically produces database documentation. As new tables are added to the database, that documentation also allows the RDBMS to check for and eliminate homonyms and synonyms.

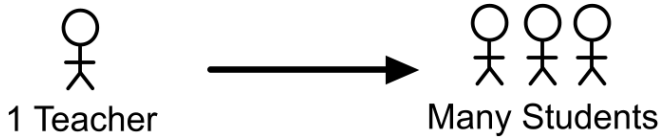
Prescribed and Recommended Textbooks/Readings

Relationships within the Relational Database

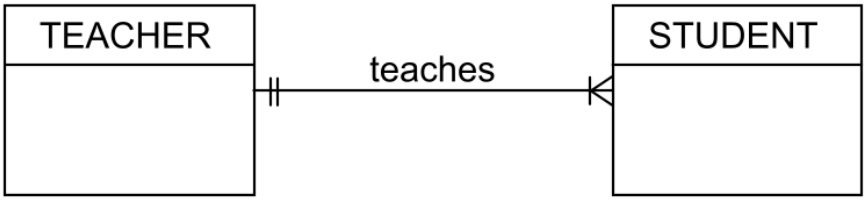
Study section 3.5 of Rob et al.

The one-to-many relationship

The one-to-many relationship is by far the most common relationship in relational databases, and is considered to be the norm in relational databases. A one-to–one relationship denotes that one entity in one table can be related to many entities in another. Consider the example from the previous chapter, where a teacher teaches many students:



This 1:M relationship can be expressed as an Entity Relationship diagram, as follows:



The above relationship will be represented in data tables as follows:

Table: STUDENT

Student_No	Name	Surname	Telephone	Email	Reg_year	Teacher_id
3045	Sam	Lakota	0112344568	sam@samail.co.za	2008	123
5623	Abe	Khumalo	0315436543	abek@email.com	2010	145
4321	John	Smith	0326548765	js@abc.co.za	2009	165
4364	Mary	Watson	0125436543	mary@abc.com	2009	145
2097	Peter	Brown	0115673456	peterb@def.co.za	2010	123
4356	John	Smith	0312345678	smithj@qwety.com	2011	123

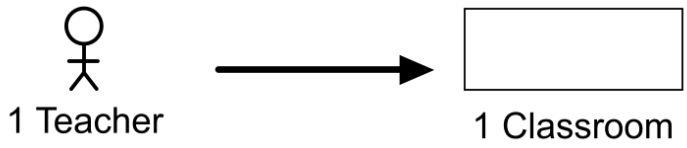
Table: TEACHER

Teacher_id	Name	Surname	Telephone	Email
123	Tim	Henks	0315556789	thenks@samail.co.za
145	Mich	Khumalo	0312324545	michk@abc.com
165	John	Harris	0326548787	jh@def.co.za

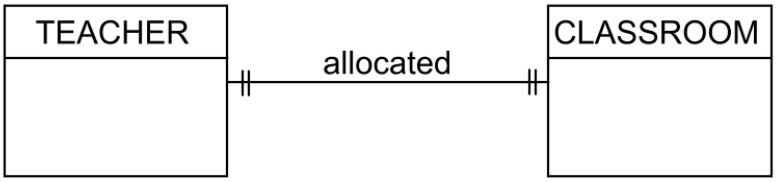
As you can see from the above tables, the teacher with Teacher_id 123, Tim Henks, teaches 3 students, which denotes a 1:M relationship.

The 1:1 Relationship

In the 1:1 relationship, one entity can be related to only one other entity, and vice versa. A typical example of this is where teaches are allocated classrooms, as shown in the previous chapter, and depicted in the diagram below:



The one-to-one relationship is modelled below:



This relationship can be implemented as shown below:

Table: TEACHER

Teacher_id	Name	Surname	Telephone	Email
123	Tim	Henks	0315556789	thenks@samail.co.za
145	Mich	Khumalo	0312324545	michk@abc.com
165	John	Harris	0326548787	jh@def.co.za

Table: CLASSROOM

Room_No	Block	Teacher_id
26	B	165
14	A	145
42	C	123

One-to-one relationships are very rare, but there are certain circumstances when they are indispensable.

The M:N Relationship

A many-to-many (M:N) relationship is not supported directly in the relational environment. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities.

Consider a typical college environment in which each STUDENT can take many CLASSes, and each CLASS can contain many STUDENTs. Each CLASS can have many STUDENTs, and each STUDENT can take many CLASSes. There can be many rows in the CLASS table for any given row in the STUDENT table, and there can be many rows in the STUDENT table for any given row in the CLASS table.

This can be implemented as follows:

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

STU_NUM	STU_LNAME	CLASS_CODE
321452	Bowser	10014
321452	Bowser	10018
321452	Bowser	10021
324257	Smithson	10014
324257	Smithson	10018
324257	Smithson	10021

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: STU_NUM

CLASS_CODE	STU_NUM	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	321452	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10014	324257	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	321452	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10018	324257	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	321452	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10021	324257	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

However, the M:N relationship should *NOT* be implemented as shown above for two reasons:

- The tables create many redundancies. As you can see, the Student_No values occur many times in the STUDENT table.
- Given the structure and contents of the two tables, the relational operations become very complex and are likely to lead to system efficiency errors and output errors.

These problems in the M:N relationship can be avoided by creating a **composite entity** (also known as a **bridge entity** or an **associative entity**). Because such a table is used to link the tables that were originally related in an M:N relationship, the composite entity structure includes – as foreign keys – *at least* the primary keys of the tables that are to be linked. There are two main options when defining a composite table's primary key: use the combination of those foreign keys or create a new primary key. The M:N student and class relationship can be correctly implemented as follows:

Table: STUDENT

Primary key: STU_NUM

Foreign key: none

STU_NUM	STU_LNAME
345678	Harris
345890	Johnson

Table: CLASS

Primary key: CLASS_CODE

Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_ROOM	CLASS_SECTION
10145	INF2B	A54	3
10146	ACC2A	B12	2
10158	BIS2A	A53	1

Table: ENROL

Primary key: CLASS_CODE + STU_NUM

Foreign key: CLASS_CODE, STU_NUM

CLASS_CODE	STU_NUM	ENROL_GRADE
10145	345678	C
10145	345890	B
10146	345678	A
10146	345890	B
10158	345678	C
10158	345890	C

The M:N relationship is implemented by introducing the ENROL class, which forms a "bridge" between the STUDENT table and the CLASS table. The ENROL table simply incorporates both tables by using their primary keys as foreign keys.

3.6 Data Redundancy Revisited

In the first chapter, you learned that data redundancy leads to data anomalies that destroy the effectiveness of the database. However, these redundancies by using common attributes that are shared by tables, called foreign keys. Although the use of foreign keys does not totally eliminate data redundancies, because the foreign key values can be repeated many times, the proper use of foreign keys *minimizes* data redundancies, thus minimizing the chance that destructive data anomalies will develop.

As important as data redundancy control is, there are times when the level of data redundancy must actually be increased to make the database serve crucial information purposes. There are also times when data redundancies *seem* to exist to preserve the historical accuracy of data.

3.7 Indexes

An index in a database table serves the same purpose as an index in a book: it helps to find things quickly and easily. If you are searching for a specific concept in a book, you could start at the beginning of the book and search sequentially, line by line and page by page, until you find what you are searching for. Clearly, this will be a very

tedious and time-consuming process. Rather than that, you will page to the index of the book and find the item in the index, and then go directly to the page it refers to.

Indexes in the relational database environment work very much like those in books. They facilitate quick and efficient data retrieval from tables. A typical database table may contain thousands of entities, and it will certainly become very inefficient and time consuming to search through every record every time a single record is required. Instead, an index is used. An index is composed of an index key and a set of pointers. The index key is the index's reference point. Each key points to the location of the data identified by the key.

Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared. For example, in a CUSTOMER table, if you declare CUS_CODE to be the primary key of the table, the DBMS automatically creates a unique index on that attribute. A unique index, as its name implies, is an index in which the index key can have only one pointer value (row) associated with it. A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (this is known as a composite index, as you have seen earlier in this chapter).

3.8 Codd's Relational Database Rules

Dr. E.F. Codd published a list of 12 rules to define a relational database system, which serves as a frame of reference for what a truly relational database should be.

- **Information** – all information in a relational database must be logically represented as column values in rows within tables.

- **Guaranteed Access** – every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.

- **Systematic Treatment of Nulls** – nulls must be represented and treated in a systematic way, independent of data type.

- **Dynamic Online Catalog Based on the Relational Model** – the metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.

- **Comprehensive Data Sublanguage** – the relational database may support many languages. However, it must support one well-defined, declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).

- **View Updating** – any view that is theoretically updatable must be updatable through the system.

- **High-Level Insert, Update, and Delete** – the database must support set-level inserts, updates, and deletes.

- **Physical Data Independence** – application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.

- **Logical data Independence** – application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).

- **Integrity Independence** – all relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.

- **Distribution Independence** – the end users and application programs are unaware and unaffected by the data location (distributed vs. local databases).

- **Non-subversion** – if the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
- **Rule Zero** – all preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational facilities exclusively to manage the database.

3.9 Summary

- The relational model enables you to view data logically rather than physically
- The logical view of the relational database is simplified by the creation of tables which are simply perceived as a two-dimensional structure composed of rows and columns.
- Keys are used to ensure that each row in a table is uniquely identifiable and establishes relationships among tables and ensures integrity of the data by preventing duplicates, for example.
- A **primary key** is an attribute, or a combination of attributes, that uniquely identifies any given row
- A **superkey** is a key that uniquely identifies each row, and can be a single attribute or a multi-attribute composite key, provided the primary key is part of that superkey.
- A **candidate key** is a superkey without unnecessary attributes, that is, it is a minimal superkey.
- A **foreign key** is a value in an attribute in a table which serves as a primary key in another table, and which serves to link the two tables.
- A relational schema is a textual representation of the database tables where each table is listed by its name followed by the list of its attributes in parentheses

- A table exhibits **entity integrity** when each primary key value is unique
- A table has **referential integrity** if it has a foreign key which matches a primary key value in the linked table
- Relational databases manipulate data via the eight relational operators: select, project, union, intersect, difference, product, join and divide
- A **data dictionary** provides a detailed description of all tables found within the user/designer-created database
- Table relationships in a database are classified as one-to-one (1:1), one-to-many (1:M) and many-to-many (M:M)
- Indexes in the relational database environment facilitate quick and efficient data retrieval from tables. An index is composed of an index key and a set of pointers. The index key is the index's reference point. Each key points to the location of the data identified by the key.
- Dr. E.F. Codd published a list of 12 rules to define a relational database system, which serves as a frame of reference for what a truly relational database should be

Revision Questions

1. What is a relation?
2. What is a relational database?
3. What is a primary key?
4. How is the term *attribute* used in the relational model? What is a more common name for *attribute*?
5. Describe the purpose of the *product* operation in a relational database.
6. What is the purpose of a system catalogue?
7. List Codd's relational database rules.

3.10 Answers to Activities

1. A relation is a two-dimensional table in which (1) the entries in the table are single-valued; (2) each column has a distinct name; (3) all of the values in a column are values of the same attribute; (4) the order of the columns is immaterial; (5) each row is distinct; and (6) the order of the rows is immaterial.

2. A relational database is a collection of relations

3. The primary key is the column or collection of columns that uniquely identifies a given row.

4. In the relational model, an attribute is a property of an entity. Attribute is another term for a column in a table. It also is commonly called a field.

5. The PRODUCT command (mathematically called the Cartesian product) is the table obtained by concatenating every row in the first table with every row in the second table.
6. A **system catalogue** contains metadata and can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. The terms *system catalogue* and *data dictionary* are often used interchangeably because the system catalogue contains all required data dictionary information.

Unit 4:

Entity Relationship (ER) Modelling

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
4.1 Introduction	• Introduce topic areas for the unit
4.2 The Entity Relationship Model (ERM)	• Know the main characteristics of entity relationship components
4.3 Relationships	• Understand how relationships between entities are defined, refined, and incorporated into the database design process
4.4 Relationship participation	• Understand how ERD components affect database design and implementation
4.5 Developing an ER Diagram	• Know that real-world database design often requires the reconciliation of conflicting goals
4.6 Summary	• Summarise topic areas covered in unit

Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

4.1 Introduction

Data modelling is the first step in database design, serving as a bridge between real-world objects and the database model that is implemented in the computer. Therefore, data modelling details are expressed graphically through entity relationship diagrams (ERDs) and are of great importance.

4.2 The Entity Relationship Model (ERM)

Prescribed and Recommended Textbooks/Readings

The Entity Relationship Model (ERM)

Study section 5.1 of Rob et al.

The ERM forms the basis of an ERD (Entity Relationship Diagram), which represents the conceptual database as viewed by the end user. ERDs depict the database's main components:

- entities
- attributes
- relationships

Because an entity represents a real-world object, the words *entity* and *object* are often used interchangeably. Three notations of ERDs will be described in this chapter:

1. Chen notation
2. Crow's Foot
3. UML notations.

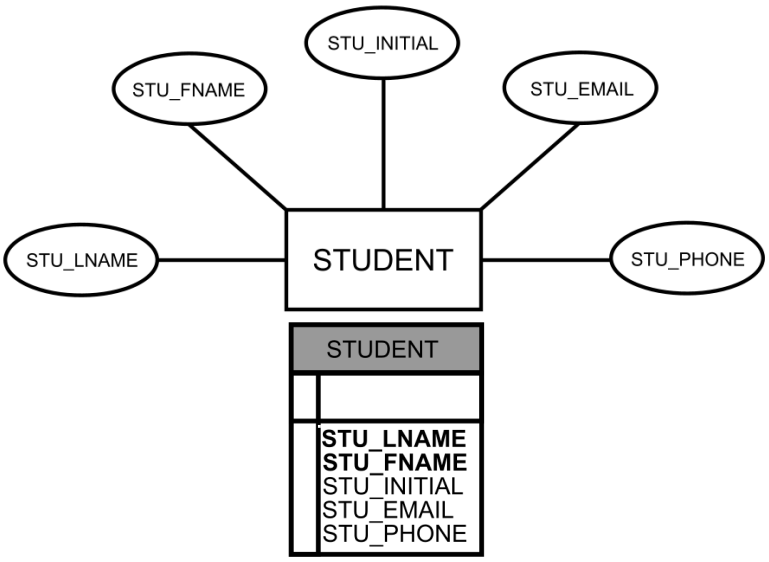
Entities

An entity is an object of interest to the end user, and object which the user wishes to store data about. The word *entity* in the ERM corresponds to a table – not to a row – in the relational environment. The ERM refers to a table row as an *entity instance* or *entity occurrence*. In both the Chen and Crow's Foot notations, an entity is represented by a rectangle containing the entity's name. The entity name, a noun, is usually written in all capital letters.

Attributes

Attributes are characteristics of entities, for example the ERD of STUDENT entity (shown below) has attributes like STU_LNAME, STU_FNAME and STU_INITIAL. In the Chen notation, attributes are represented by ovals and are connected to the entity rectangle with a line. In the Crow's Foot notation, the attributes are written in the attribute box below the entity rectangle.

Chen notation:



Crow's foot notation:

Required and Optional Attributes

A **required attribute** is an attribute that must have a value, that is, the attribute cannot be left empty. For example, in the STUDENT Crow's Foot notation, STU_LNAME and STU_FNAME are two required attributes indicated in boldface.

An **optional attribute** is an attribute that does not require a value, that is, the attribute can be left empty. For example, in the same STUDENT Crow's Foot notation, STU_INITIAL, STU_EMAIL and STU_PHONE are considered optional attributes.

Domains

A domain is the set of possible values for a given attribute. For example, the domain for the gender attribute consists of only two possibilities: M or F. The domain for the grade point average (GPA) attribute is written (0,4) where the lowest possible GPA value is 0 and the highest possible value is 4.

Identifiers (Primary Keys)

The ERM uses **identifiers**, one or more attributes, to uniquely identify each entity instance. In relational models, these identifiers are mapped to primary keys (PKs) in tables. Identifiers are underlined in the ERD. For example, a CAR entity may be represented by:

CAR(**CAR_VIN**, MOD_CODE, CAR_YEAR, CAR_COLOR)

(Each car is identified by a unique vehicle identification number, or CAR_VIN.)

Composite Identifiers

Ideally, an entity identifier is composed of only a single attribute. However, it is possible to use the **composite identifier**, a primary key composed of more than one attribute.

Referring to the table below, we see that CLASS_CODE is the CLASS table's designated primary key.

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

However, combining CRS_CODE and CLASS_SECTION and deleting CLASS_CODE from the CLASS table would make this a candidate key (CRS_CODE and CLASS_SECTION combined), and becomes an acceptable composite primary key.

CLASS (**CRS_CODE**, **CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

Composite and Simple Attributes

A **composite attribute** is an attribute that can be further subdivided to yield additional attributes. For example, the attribute ADDRESS can be further subdivided into street, city, state, zip code. A composite attribute is not the same as a composite key.

A **simple attribute** is an attribute that cannot be subdivided, such as, age, sex, and marital status.

Single-Valued Attributes

A **single-valued attribute** is an attribute that can have only a single value. For example, a person can have only one identification number (ID) or a manufactured product part can have only one serial number.

Keep in mind that a single-valued attribute is not necessarily a simple attribute. For instance, a part's serial number, such as SE-08-02-189935, is single-valued, but it is in itself a composite attribute because it can be subdivided

into the region in which the part was produced (SE), the plant within that region (08), the shift within the plant (02), and the part number (189935).

Multivalued Attributes

Multivalued attributes are attributes that can have many values. For instance, a person may have several college degrees, and a household may have several different phones, each with its own number.

Implementing Multivalued Attributes

Multivalued attributes are a bit tricky to deal with. As an example, consider the following table:

ID_NUMBER	NAME	SURNAME	DEGREE
54321	Tom	Smith	B. Comm

The above table will be fine if Tom Smith has only one qualification, but what will be the case if he had another qualification such as an MBA too? In such a case, the designer must decide on one of two possible courses of action:

1. Within the original entity, create several new attributes, one for each of the original multivalued attribute's components. For example, we can create multiple attributes for the student's qualification, such as: DEGREE_ONE, DEGREE_TWO, DEGREE_THREE and so forth. The result of this plan would be:

ID_NUM	NAME	SURNAME	DEGREE_ONE	DEGREE_TWO	DEGREE_THREE
54321	Tom	Smith	B. Comm	MBA	B.Sc

   As you may have well guessed by now, this is not a very effective solution because a person might have more than three degrees, and in such a case the entire table would have to be modified to include another column called DEGREE_FOUR. In fact, it is impossible from the outset to anticipate how many qualifications people will have, and so it is impossible for the database designer to design a table accurately unless the maximum number of attributes is fixed.

2. A second, more practical solution would be to create a new entity (table) composed of the original multivalued attribute's components. This new entity allows the designer to define any number of degrees for different people. Then, the new DEGREES entity is related to the original PERSON entity in a 1:M relationship, as shown below:

ID_NUMBER	NAME	SURNAME	DEGREE
54321	Tom	Smith	B. Comm

PERSON_ID	DEGREE
54321	B.Comm
54321	MBA
54321	B.Sc.

Using the approach of solution 2, you are able to assign as many attributes, such as degrees, as necessary without having to change the table structure. Solution 2 is the preferred way to deal with multivalued attributes.

Derived Attributes

A **derived attribute** is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB.

Derived attributes are sometimes referred to as *computed attributes*. A derived attribute computation can be as simple as adding two attribute values located on the same row, or it can be the result of aggregating the sum of values located on many table rows from the same table or from a different table).

4.3 Relationships

A relationship is an association between entities. The entities that participate in a relationship are also known as **participants**, and each relationship is identified by a name that describes the relationship. A relationship name is an active or passive verb. For example, a STUDENT *takes* a CLASS, a PROFESSOR *teaches* a CLASS, a DEPARTMENT *employs* a PROFESSOR, a DIVISION *is managed by* an EMPLOYEE, and an AIRCRAFT *is flown by* a CREW.

Relationships between entities always operate bi-directionally. To define the relationship between the entities named CUSTOMER and INVOICE, you would specify that:

- A CUSTOMER may generate many INVOICEs.
- Each INVOICE is generated by one CUSTOMER.

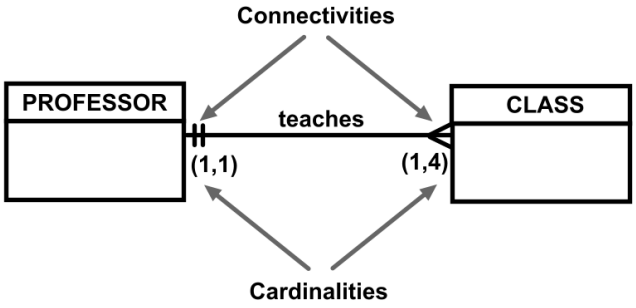
You can easily see that this relationship is classified as a 1:M relationship.

Connectivity and Cardinality

Connectivity is a term used to describe the relationship classification; that is, it is used to describe whether the relationship is a one-to-one, one-to-many or many-to-many relationship.

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In other words, it describes how many of one entity in a relationship will be associated with the other.

In entity relationship diagrams, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (*x,y*). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities. Look at the diagram below:



In the above diagram, the cardinality (1,4) written next to the CLASS entity in the "PROFESSOR teaches CLASS" relationship indicates that each professor teaches up to four classes, which means that the PROFESSOR table's primary key value occurs at least once and no more than four times as foreign key values in the CLASS table. If the cardinality had been written as (1,N), there would be no upper limit to the number of classes a professor might teach.

Similarly, the cardinality (1,1) written next to the PROFESSOR entity indicates that each class is taught by one and only one professor. That is, each CLASS entity occurrence is associated with one and only one entity occurrence in PROFESSOR.

Connectivities and cardinalities are established by very concise statements known as business rules, which derived from a precise and detailed description of an organization's data environment, also establish the ERM's entities, attributes, relationships, connectivities, cardinalities, and constraints.

Existence Dependence

An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with *another* related entity occurrence. In other words, an entity is existence-dependent if it has a mandatory foreign key – that is, a foreign key attribute that cannot be null. For example, if an employee wants to claim one or more dependents for tax-withholding purposes, the following relationship would be appropriate:

EMPLOYEE claims DEPENDENT

In the above case, the DEPENDENT entity is clearly existence-dependent on the EMPLOYEE entity because it is impossible for the dependent to exist apart from the EMPLOYEE in the company database. A dependant is always linked to an employee of the company.

If an entity can exist on its own without any other entities, it is said to be **existence-independent**. That entity is referred to as a **strong entity** or **regular entity**.

For example, suppose that the XYZ Corporation uses parts to produce products, and those parts are purchased from other companies. In such a case, a part cannot exist in the company database without a supplier, so it is existence-dependent on the supplier table. On the other hand, a supplier may be listed on XYZ Corporation's database without ever having supplied XYZ with any parts. This is because the suppliers are existence-independent.

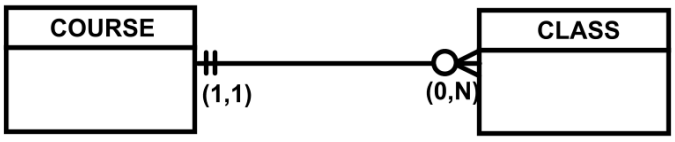
4.4 Relationship participation

Relationships are bidirectional, meaning they operate in both directions. For example, if COURSE is related to CLASS, then by definition, CLASS is related to COURSE. In bidirectional relationships, it is necessary to determine the connectivity of the relationship from COURSE to CLASS and the connectivity of the relationship from CLASS to COURSE. The specific maximum and minimum cardinalities must be determined in each direction for the relationship.

Participation in an entity relationship is either a) optional or b) mandatory.

Optional participation

In optional participation, one entity occurrence does not require a corresponding entity occurrence in a particular relationship. For example, in the "COURSE generates CLASS" relationship, at least some courses do not generate a class. An entity occurrence (row) in the COURSE table does not necessarily require the existence of a corresponding entity occurrence in the CLASS table. Therefore, the CLASS entity is considered to be *optional* to the COURSE entity.



In Crow's Foot notation, an optional relationship between entities is shown by drawing a small circle (O) on the side of the optional entity. The existence of an *optional entity* indicates that the minimum cardinality is 0 for the optional entity.

Mandatory participation

In mandatory participation one entity occurrence *requires* a corresponding entity occurrence in a particular relationship.

The Crow's Foot notation supports the following connectivity and participation combinations:

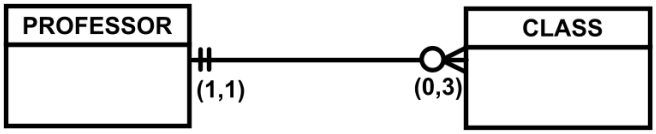
CROW'S FOOT SYMBOL	CARDINALITY	COMMENT
	(0,N)	Zero or many. Many side is optional.
	(1,N)	One or many. Many side is mandatory.
	(1,1)	One and only one. 1 side is mandatory.
	(0,1)	Zero or one. 1 side is optional.

ider the following examples:

Example 1

Suppose that Tiny College employs some professors who conduct research without teaching classes.
If you examine the "PROFESSOR teaches CLASS" relationship, it is quite possible for a PROFESSOR not to teach a CLASS. Therefore, CLASS is *optional* to PROFESSOR.

On the other hand, a CLASS must be taught by a PROFESSOR. Therefore, PROFESSOR is *mandatory* to CLASS. This will be depicted in a crow's foot diagram as follows:



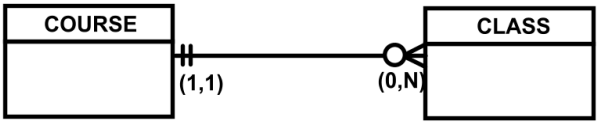
In the above ERD, the cardinality next to CLASS is (0,3) indicating that a professor may teach no classes at all or as many as three classes. And each CLASS table row will reference one and only one PROFESSOR row – assuming each class is taught by one and only one professor – represented by the (1,1) cardinality next to the PROFESSOR table.

It is vitally important to clearly understand the concepts of mandatory and optional participation.

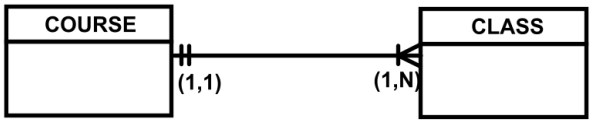
Example 2:

Suppose that Tiny College offers several courses, and each course has several classes. There are two possible scenarios for this:

1. A course may or may not have any classes running; that is CLASS is optional. This will be depicted as follows:



2. A course has to have at least one class running; that is, class is mandatory. This will be depicted as follows:



Analysing the relationships, it is easy to see that in both cases a CLASS cannot exist without a COURSE. Therefore, you can conclude that the COURSE entity is *mandatory* in the relationship.

4.5 Developing an ER Diagram

The process of database design is an iterative rather than a linear or sequential process and building any ERD usually involves the following activities:

- Create a detailed narrative of the organization's description of operations.
- Identify the business rules based on the description of operations.
- Identify the main entities and relationships from the business rules.
- Develop the initial ERD.
- Identify the attributes and primary keys that adequately describe the entities.
- Revise and review the ERD.

During the review process, additional objects, attributes, and relationships will be uncovered and the basic ERM will be modified to incorporate the newly discovered ER components. Another round of reviews might yield additional components or clarification of the existing diagram. This process is repeated until it is agreed that the ERD is a fair representation of the organization's activities and functions.

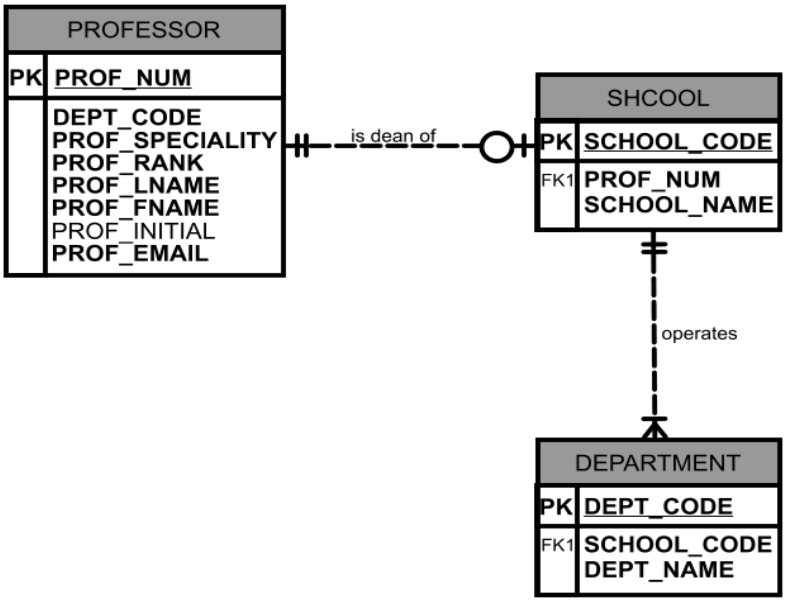
During the design process, not just interviews to define entities, attributes, and relationships, but also a good amount of information should be gathered by examining the business forms and reports that an organization uses in its daily operations.

To illustrate the use of the iterative process that ultimately yields a workable ERD, let's start with an initial interview with the Tiny College administrators. The interview process yields the following business rules:

1. Tiny College is divided into several schools. Each school is administered by a dean who is a professor. Each professor can be the dean of only one school, and a professor is not required to be the dean of any school.

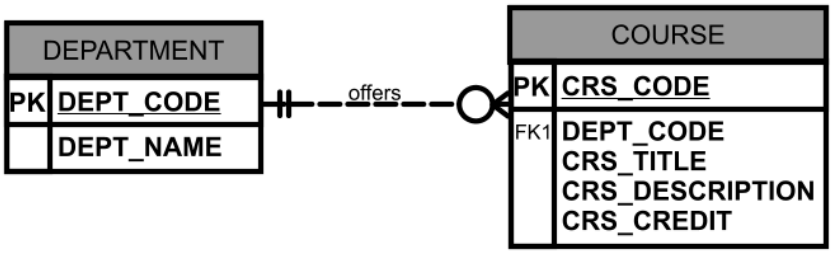
2. Therefore, a 1:1 relationship exists between PROFESSOR and SCHOOL. The cardinality can be expressed by writing (1,1) next to the entity PROFESSOR and (0,1) next to the entity SCHOOL.

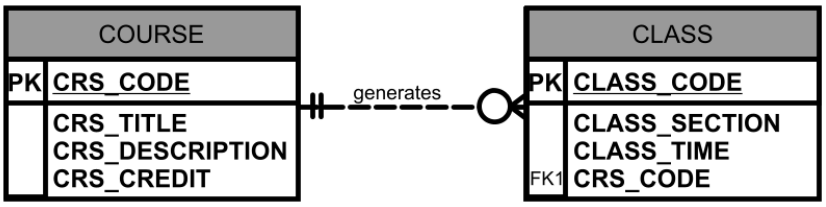
3. Each school comprises several departments. The smallest number of departments operated by a school is one, and the largest number of departments is indeterminate (N). Each department belongs to only a single school; thus, the cardinality is expressed by (1,1). Thus, the minimum and maximum number of schools that a department belongs to is one.



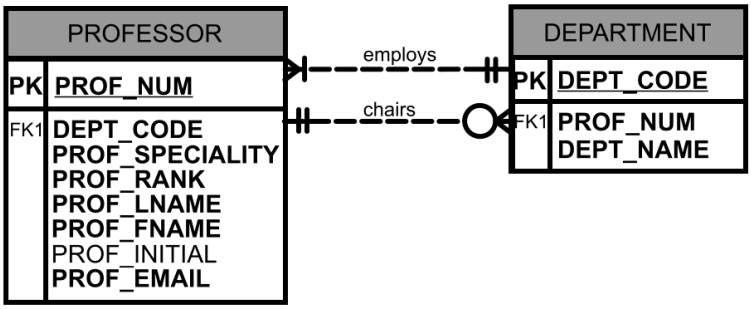
4. Each department may offer courses. Tiny College had some departments that were classified as "research only", those departments would not offer courses; therefore, the COURSE entity would be optional to the DEPARTMENT entity.

5. A CLASS is a section of a COURSE. That is, a department may offer several sections (classes) of the same database course. A 1:M relationship exists between COURSE and CLASS. However, CLASS is optional to COURSE, because a course may exist in the course catalogue even when it is not offered as a class in a current class schedule.

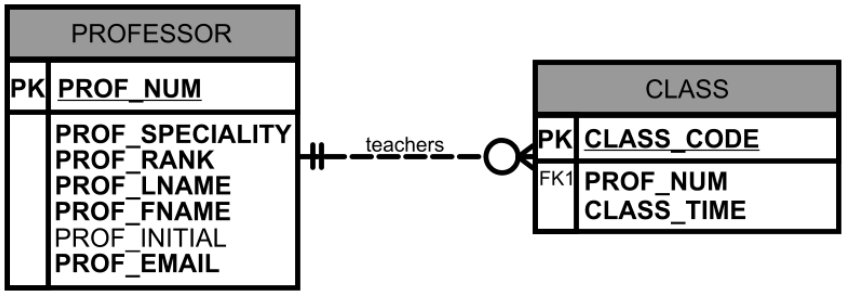


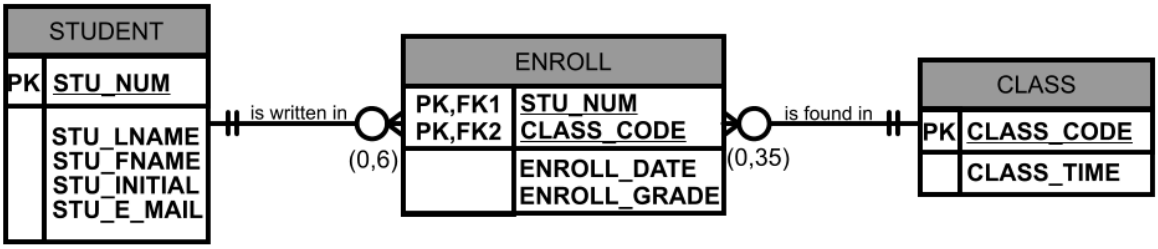


6. Each department should have one or more professors assigned to it. One and only one of those professors chairs the department, and no professor is required to accept the chair position. Therefore, DEPARTMENT is optional to PROFESSOR in the "chairs" relationship.

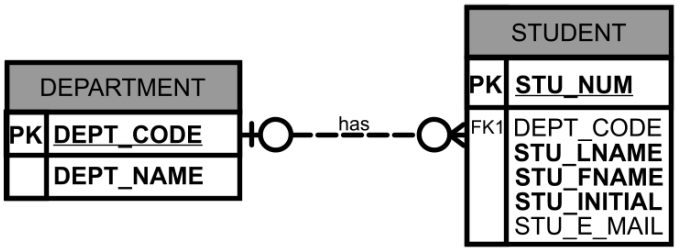


7. Each professor may teach up to four classes; each class is a section of a course. A professor may also be on a research contract and teach no classes at all.
8. A student may enrol in several classes but takes each class only once during any given enrolment period. Thus creating an M:N relationship between STUDENT and CLASS. This M:N relationship must be divided into two 1:M relationships through the use of the ENROLL entity. However, the ENROLL entity is weak, as its existence-dependent and its (composite) PK is composed of the PK's of the STUDENT and CLASS entities.

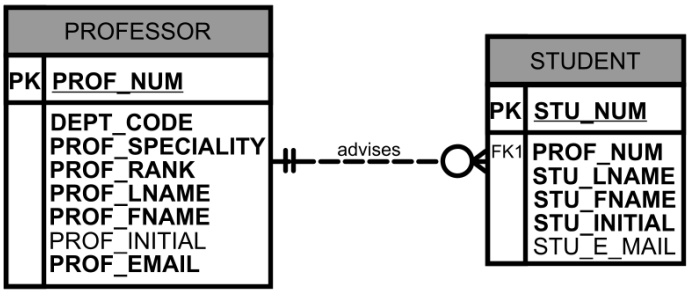




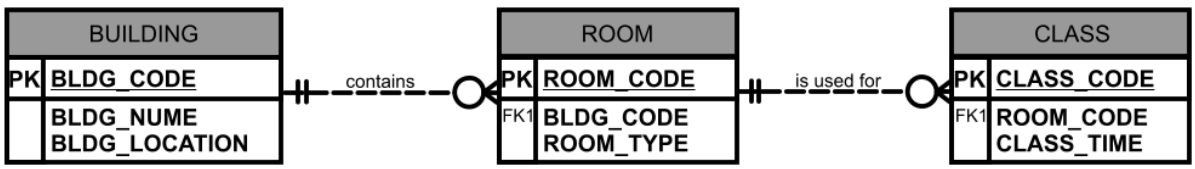
9. Each department has several (or many) students whose major is offered by that department. However, each student has only a single major and is, therefore, associated with a single department.



10. Each student has an advisor in his or her department; each advisor counsels several students. An advisor is also a professor, but not all professors advise students. Therefore, STUDENT is optional to PROFESSOR in the "PROFESSOR advises STUDENT" relationship.



11. The CLASS entity contains a ROOM_CODE attribute. Given the naming conventions, ROOM_CODE is an FK to another entity, ROOM. In turn, each room is located in a building.

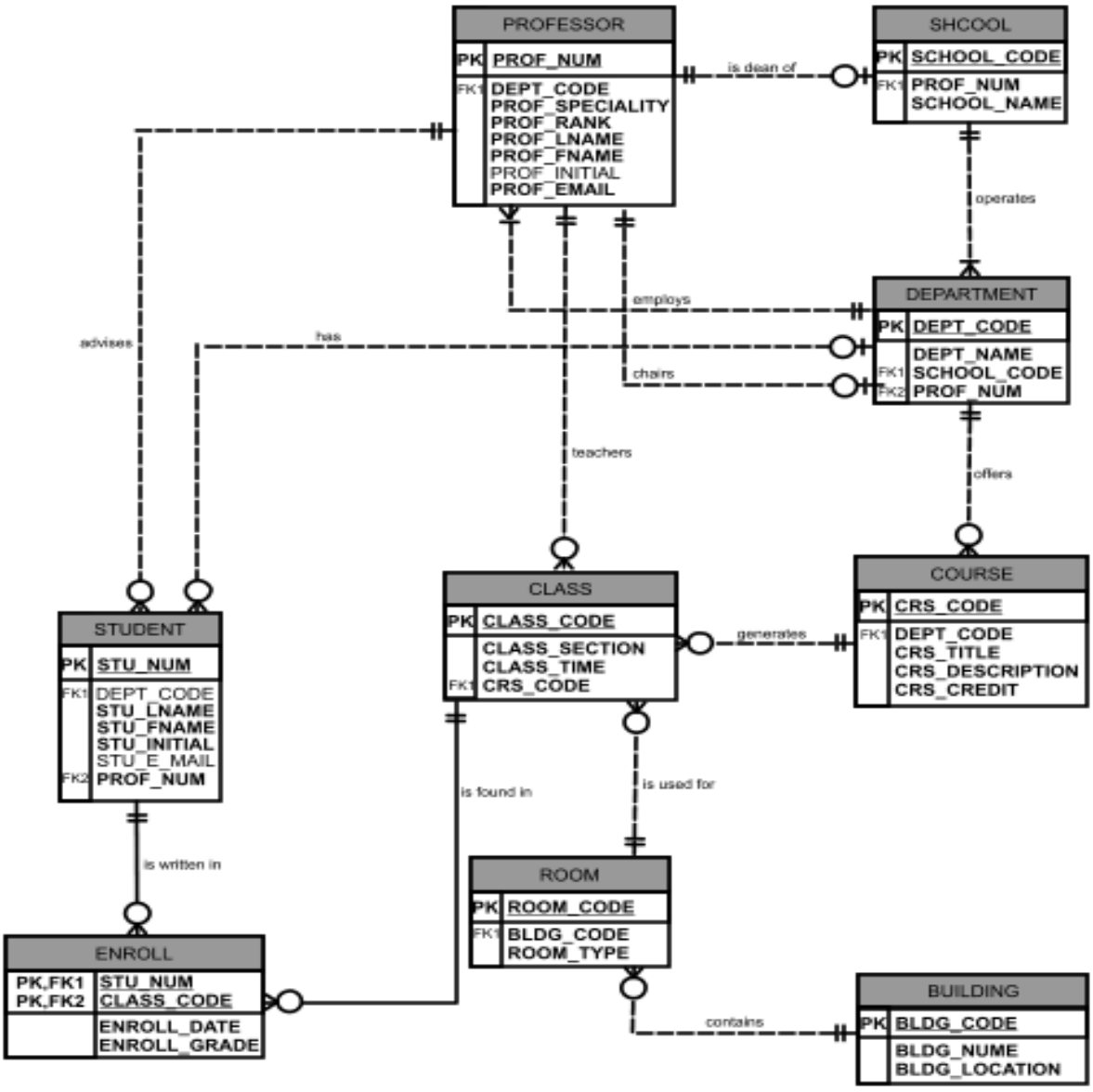


Once the relevant entities are discovered, the initial set of relationships can be defined amongst them. Next, is to describe the entity attributes.

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrols in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS
ENROL is the composite entity that implements the M:N relationship “STUDENT enrols in CLASS”			

Components of the ERM

The completed ERD for Tiny College is on the next page.


PROFESSOR
PK PROF_NUM
FK1 DEPT_CODE
PROF_SPECIALITY
PROF_RANK
PROF_LNAME
PROF_FNAME
PROF_INITIAL
PROF_EMAIL
is dean of
SHCOOL
PK SCHOOL_CODE
FK1 PROF_NUM
SCHOOL_NAME
operates
employs
DEPARTMENT
PK DEPT_CODE
DEPT_NAME
FK1 SCHOOL_CODE
FK2 PROF_NUM
advises
has
chairs
teachers
offers
CLASS
PK CLASS_CODE
CLASS_SECTION
CLASS_TIME
FK1 CRS_CODE
generates
COURSE
PK CRS_CODE
FK1 DEPT_CODE
CRS_TITLE
CRS_DESCRIPTION
CRS_CREDIT
STUDENT
PK STU_NUM
FK1 DEPT_CODE
STU_LNAME
STU_FNAME
STU_INITIAL
STU_E_MAIL
FK2 PROF_NUM
is found in
is used for
is written in
ROOM
PK ROOM_CODE
FK1 BLDG_CODE
ROOM_TYPE
ENROLL
PK,FK1 STU_NUM
PK,FK2 CLASS_CODE
ENROLL_DATE
ENROLL_GRADE
contains
BUILDING
PK BLDG_CODE
BLDG_NUME
BLDG_LOCATION

Database Design Challenges: Conflicting Goals

Database designers often must make design compromises that are triggered by conflicting goals, such as:

- Design standards (design elegance) - The database design must conform to design standards that guide you to developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur. Without design standards, it is nearly impossible to formulate a proper design process, to evaluate an existing design, or to trace the likely logical impact of changes in design.

- Processing speed – High processing speeds are often a top priority in database design in many organizations. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships.
- Information requirements – Complex information requirements may dictate data transformations, and they may expand the number of entities and attributes within the design. Therefore, the database may have to sacrifice some of its "clean" design structures and/or some of its high transaction speed to ensure maximum information generation.

You are quite likely to discover that even the best design process produces an ERD that requires further changes mandated by operational requirements. Such changes should not discourage you from using the process. ER modelling is essential in the development of a sound design that is capable of meeting the demands of adjustment and growth. Using ERDs yields perhaps the richest bonus of all: a thorough understanding of how an organization really functions.

Your job as a database designer is to use your professional judgement to yield a solution that meets the requirements imposed by business rules, processing requirements, and basic design principles.

Finally, documentation not only helps you stay on track during the design process, but also enables you to pick up the design thread when the time comes to modify the design. Design and implementation means "PUT ALL DESIGN ACTIVITIES IN WRITING". The development of organizational documentation standards is a very important aspect of ensuring data compatibility and coherence.

4.6 Summary

- Data modelling is the first step in database design, serving as a bridge between real-world objects and the database model that is implemented in the computer.
- The Entity Relationship Model (ERM) forms the basis of an Entity Relationship Diagram (ERD)
- ERDs depict the database's main components: entities, attributes and relationships.
- An entity is an object of interest to the end user, and object which the user wishes to store data about.
- Attributes are characteristics of entities.
- A relationship is an association between entities.
- The following are popular ERD notations:

- Chen notation
- Crow's Foot
- UML notations

- Connectivity is a term used to describe the relationship classification; that is, it is used to describe whether the relationship is a one-to-one, one-to-many or many-to-many relationship.
- Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity.
- An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with *another* related entity occurrence.
- If an entity can exist on its own without any other entities, it is said to be **existence-independent**.
- A **relationship degree** indicates the number of entities or participants associated with a relationship.

Revision Questions

1. What is an entity?
2. What are the components of an ERD?
3. Differentiate between required attributes and optional attributes. Provide an example of each.
4. What is a composite identifier?
5. Describe the best way to implement a multi-valued entity, with an example.
6. Differentiate between connectivity and cardinality.
7. Given the following business rules, create the appropriate Crow's Foot ERD:
   a. A company operates many departments
   b. Each department employs one or more employees
   c. Each employee may or may not have one or more dependents
   d. Each employee may or may not have an employment history

4.7 Answers to Activities

1. An entity is an object of interest to the end user, and object which the user wishes to store data about. The word *entity* in the ERM corresponds to a table – not to a row – in the relational environment.

2. ERDs depict the following components:
   - entities
   - attributes
   - relationships

3. A **required attribute** is an attribute that must have a value, that is, the attribute cannot be left empty. For example, in a STUDENT table, surname and first name are two required attributes indicated in boldface.
   An **optional attribute** is an attribute that does not require a value, that is, the attribute can be left empty. For example, in the STUDENT table, telephone number, email and cellphone may be left out, and are hence considered optional attributes.

4. A composite identifier is a primary key composed of more than one attribute

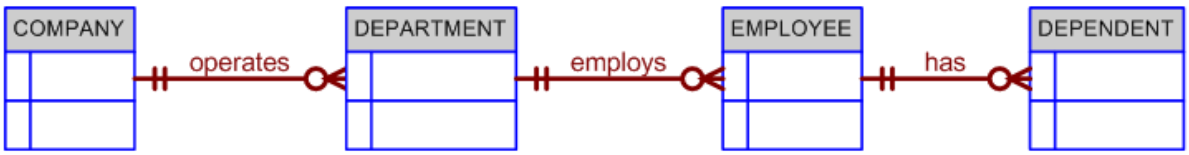
5. A practical solution would be to create a new entity (table) composed of the original multivalued attribute's components. This new entity allows the designer to define any number of degrees for different people. Then, the new DEGREES entity is related to the original PERSON entity in a 1:M relationship, as shown below:

ID_NUMBER	NAME	SURNAME	DEGREE
54321	Tom	Smith	B. Comm

PERSON_ID	DEGREE
54321	B.Comm
54321	MBA
54321	B.Sc.

6. **Connectivity** is a term used to describe the relationship classification; that is, it is used to describe whether the relationship is a one-to-one, one-to-many or many-to-many relationship.
   Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In other words, it describes how many of one entity in a relationship will be associated with the other.

7. The ERD's for all four scenarios may be combined as follows:



Unit 5:

Normalization

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
5.1 An introduction to normalisation	• Understand what normalisation is and what role it plays in the database design process
5.2 The need for normalisation	• Know about the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
5.3 The normalisation process	• Know how normal forms can be transformed from lower normal forms to higher normal forms
5.4 Adding further improvements to table designs	• Understand that normalisation and ER modelling are used concurrently to produce a good database design
5.5 Summary	• Summarise topic areas covered in unit

Prescribed /Recommended Textbooks/Readings

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

5.1 An introduction to normalization

Relational databases were developed to address many of the problems associated with previous data storage mechanisms such as data files, which had numerous shortcomings like data redundancy and data anomalies.

A well-designed relational database will be far superior to a traditional file system, but one that is poorly designed will certainly have the same issues as a file system. A poorly-designed database will not allow the relational database management system (RDBMS) to demonstrate the true power of its data-handling capabilities. Hence it is necessary to create tables which are well-structured.

The way to ensure that tables in a database are well-structured, thus eliminating data redundancies and anomalies, is via the process of normalization. Normalization is described as a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.

Normalization is implemented through a series of progressive steps called normal forms which are known as:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)

The objective of normalization is to ensure that each table in the database conforms to the concept of well-formed relations - that is, each table must have the following characteristics:

1. Each table represents a single subject. For example, a course table will contain only data that directly pertain to courses. Similarly, a student table will contain only student data.
2. No data item will be *unnecessarily* stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that any data which needs to be updated, is updated in only one place.
3. All nonprime attributes in a table are dependent on the primary key - the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data are uniquely identifiable by a primary key value.
4. Each table is free of insertion, update, or deletion anomalies. This is to ensure the integrity and consistency of the data.

5.2 The need for normalization

Prescribed and Recommended Textbooks/Readings

The need for normalization

Study section 7.2 of Rob et al carefully

In the previous chapter you studied entity relationship modelling, which offers an effective way to design good databases. ER modelling, however, cannot guarantee that the databases that are designed will have well-structured tables which are free of data redundancies and anomalies; ER modelling needs to be used in conjunction with normalization in order to produce well-structured tables.

Normalization can be used to design new databases or to rectify existing ones which are structurally corrupt. In both cases, the normalization process is the same, and is described in this chapter. In this chapter we will explain the concept as well as the process of normalization by means of the example which appears in section 7.2 of Rob et al.

The example used is that of a software development company that manages several projects. Each project has its own project number, project name, employees assigned to it, and so on. Each employee has an employee number, name, and job classification, such as engineer or computer technician.

The company charges its clients by billing the hours spent on each contract. The hourly billing rate is dependent on the employee's position. For example, one hour of computer technician time is billed at a different rate than one hour of engineer time. Periodically, a report is generated that contains the information displayed in the following table:

Table: PROJECTS

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.0
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35,75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.0
18	Amber Wave	114	Annelise Jones	Applications Designer	40.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	40.4
		113	Delbert K. Joenbrood	Applications Designer	40.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.07	22.0
		106	William Smithfield	Programmer	35.75	12.0
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.0
		101	John G. News	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	40.10	33.1

		100	Ralph B. Washington	Systems Analyst	96.75	23.6
		110	James J. Frommer	General Support	10.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Looking at the above table, it might make sense from a spread sheet perspective; however, from a relational database perspective, it does not conform to proper relational database structures, nor does it handle data very well. The table has the following deficiencies:

1. The project number (PROJ_NUM) is apparently intended to be a primary key or at least a part of a PK, but it contains nulls. As you know from the previous chapters, a primary key can never have null values.
2. The table entries invite data inconsistencies. For example, the JOB_CLASS value "Elect. Engineer" might be entered as "Elect.Eng.", "El. Eng.", or "EE". In the same way, people's names can be spelt wrongly, possibly resulting in many different entries per person.

3. The table has data redundancies. Those data redundancies yield the following anomalies:
    a. *Update anomalies*. Employee number 105 (Alice K. Johnson) appears many times in the table. If we had to modify Alice's JOB_CLASS, it would require many alterations, one for each time EMP_NUM = 105 appears in the table.
    b. *Insertion anomalies*. Based on the existing structure of the table, each employee has to be assigned to a project; in other words, an employee cannot exist without a project. If the employee is not yet assigned to any project, a phantom project must be created to complete the employee data entry.
    c. *Deletion anomalies*. Suppose that only one employee is associated with a given project. If that employee leaves the company and the employee data are deleted, the project information will also be deleted. To prevent the loss of the project information, a fictitious, "placeholder" employee must be created just to save the project information.

4. The table involves unnecessary repetition of information. For example, if employee number 101 (John G. News) is moved to project 22 (Rolling Tide) then each of the following entries will have to be made unnecessarily: PROJ_NAME, EMP_NAME, and CHG_HOUR. Imagine how inefficient this would be if there are 200 or 300 such entries!

5.3 The normalisation process

In this section you will learn how to perform normalization practically by normalizing the above table. Normalization starts at the first normal form, and progresses to the second and then the third normal form. The basic characteristics of each normal form is shown in the table below:

Normal form	Characteristic
First normal form (1NF)	Table format, no repeating groups, PK identified
Second normal form (2NF)	1NF and no partial dependencies
Third normal form (3NF)	2NF and no transitive dependencies

In order to understand the process of normalization, there are a few terms which you need to re-familiarize yourself with:

1. *Superkey*: A superkey is an attribute or set of attributes that uniquely identifies rows within a table; in other words, each row is always guaranteed to have a unique superkey.
2. *Candidate key*: a candidate key is a minimal (irreducible) superkey.
3. *Functional dependence*: The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B.*

   Example: PROJ_NUM → PROJ_NAME

   (read as *"*PROJ_NUM functionally determines PROJ_NAME*"*)

   In this case, the attribute PROJ_NUM is known as the "**determinant***"* attribute, and the attribute PROJ_NAME is known as the *"***dependent***"* attribute.
4. *Partial dependency*: A partial dependency exists when there is a functional dependence in which the determinant is only part of the primary key (remember we are assuming there is only one candidate key). For example:

   If (A, B) → (C,D) and B → C and (A, B) is the primary key, then the functional dependence B → C is a partial dependency because only part of the primary key (B) is needed to determine the value of C. Partial dependencies tend to be rather straightforward and easy to identify.
5. *Transitive dependency*: A transitive dependency exists when there are functional dependencies such that:

   $$X \rightarrow Y \text{ and } Y \rightarrow Z$$

   And X is the primary key. In that case, the dependency X → Z is a transitive dependency because X determines the value of Z via Y. Unlike partial dependencies, transitive dependencies are more difficult to identify.

Conversion to first normal form

A table is in the first normal form (1NF) when:

1. All of the key attributes are defined.
2. There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
3. All attributes are dependent on the primary key.

To convert a table into 1NF, you will need to follow 3 steps, as shown below.

Step 1: Eliminate the repeating groups

A repeating group is a group of multiple entries of the same type can exist for any *single* key attribute occurrence. For example, a single project number can reference a group of rows; project number 18 references 4 rows, while project number 22 references five rows.

A relational table must never contain repeating groups; if they do exist, they must be eliminated. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

This is shown in the table below:

Table: PROJECTS

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.0
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35,75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.0
18	Amber Wave	114	Annelise Jones	Applications Designer	40.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras	Systems Analyst	96.75	32.4

18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	40.4
22	Rolling Tide	113	Delbert K. Joenbrood	Applications Designer	40.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.07	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.0
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.0
25	Starflight	101	John G. News	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	40.10	33.1
25	Starflight	100	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	110	James J. Frommer	General Support	10.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

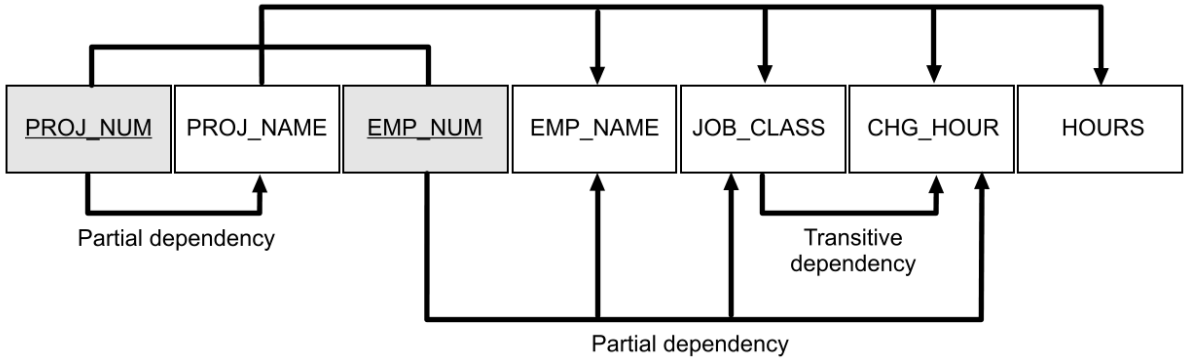
Step 2: Identify the primary key

It is clear that in the above table PROJ_NUM cannot serve as a primary key because it is not unique. As a result, we need to select a primary key which is a combination of the PROJ_NUM and at least one other attribute – in this case, the EMP_NUM attribute is ideal because each row has a unique combination of PROJ_NUM and EMP_NUM. As an example, the PROJ_NUM =25 and EMP_NUM = 110 will refer to EMP_NAME = James J. Frommer.

Step 3: Identify all dependencies

In the table above, all other attributes are dependent on the primary key which is a combination of the PROJ_NUM and EMP_NUM attributes. This is not the only dependency that exists in the table; there are other dependencies, as can be seen in the dependency diagram below:

1NF Dependency diagram



As you can see, dependency diagrams are very useful because they offer you a bird's eye view of the dependencies between the various attributes in the table. Note the following about the diagram:

1. The primary key attributes are bold, underlined, and shaded in a different colour.
2. The arrows above the attributes indicate all desirable dependencies, that is, dependencies that are based on the primary key. In this case, note that the entity's attributes are dependent on the combination of PROJ_NUM and EMP_NUM.
3. The arrows below the dependency diagram indicate the dependencies which need to be eliminated. Two types of such dependencies exist:
    a. *Partial dependencies*. You need to know only the PROJ_NUM to determine the PROJ_NAME; that is, the PROJ_NAME is dependent on only part of the primary key. And you need to know only the EMP_NUM to find the EMP_NAME, the JOB_CLASS, and the CHG_HOUR. As explained earlier, dependency based on only a part of a composite primary key is a partial dependency.
    b. *Transitive dependencies*. Note that CHG_HOUR is dependent on JOB_CLASS. Because neither CHG_HOUR nor JOB_CLASS is a prime attribute - that is, neither attribute is at least part of a key - the condition is a transitive dependency. In other words, a transitive dependency is a dependency of one nonprime attribute on another nonprime attribute. The problem with transitive dependencies is that they still yield data anomalies.

We mentioned at the beginning of this section that a table is in the first normal form (1NF) when:

1. All of the key attributes are defined.
2. There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
3. All attributes are dependent on the primary key.

Based on the above criteria, the table is now in the first normal form. Next we will progress to the second normal form.

Conversion to second normal form

When a table has a composite primary key like the table in the previous section, then it needs to be converted to the second normal form. Remember that a table in 1NF which has a single-attribute primary key is automatically in 2NF; in other words, there is no need to do any work on it to convert it to 2NF.

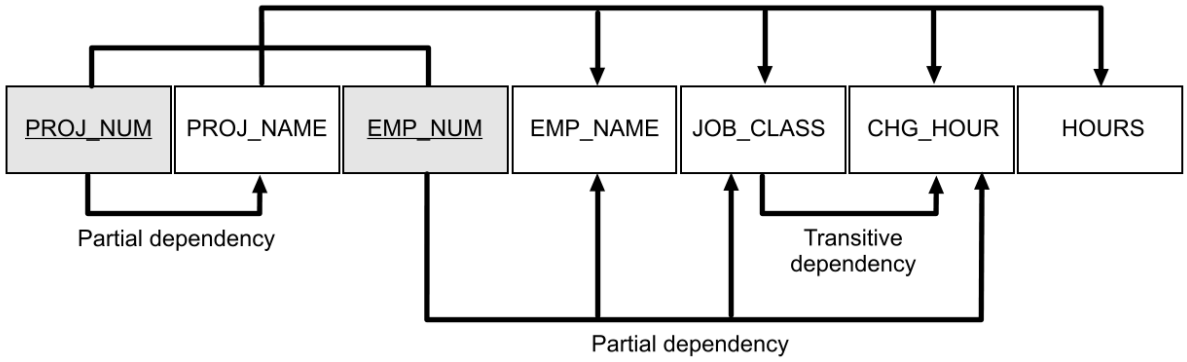
A table is in **second normal form** (**2NF**) when:

- It is in 1NF.
- It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key. Note that it is still possible for a table in 2NF to exhibit transitive dependency; that is, the primary key may rely on one or more nonprime attributes to functionally determine other nonprime attributes, as is indicated by a functional dependence among the nonprime attributes.

To convert a table from 1NF to 2NF you need to follow two steps.

Step 1: Make new tables to eliminate partial dependencies

Let's take another look at the dependency diagram for the PROJECTS table:



The table has two partial dependencies:

1. PROJ_NAME is dependent on PROJ_NUM
2. EMP_NAME, JOB_CLASS and CHG_HOUR are dependent on EMP_NUM

We need to remove these partial dependencies. To do this is simple: we need to create a new table for each partial dependency. We start by taking each attribute in the primary key and making it the primary key of a new table, but at the same time keep the original table. We will thus have three primary keys which will belong to three tables:

PROJ_NUM

EMP_NUM

And the original composite key from the PROJECTS table:

PROJ_NUM EMP_NUM

As you can see, we've written each component of the composite primary key in a separate line, and then we've placed the component primary key in the last line. This is the conventional way of doing it.

Step 2: Reassign dependent attributes to the appropriate table

Now that we've determined the primary keys for the new tables, we need to create the actual tables and determine which attributes should go into each table. This step is also quite simple: all that is required is to study the dependency diagram and see which attribute are dependent on each primary key. Looking at the diagram, we can describe the following relational schemas:

PROJECT(**PROJ_NUM**, PROJ_NAME)

EMPLOYEE(**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

But there is one attribute that does not depend on either EMP_NUM or PROJ_NUM; that attribute is ASSIGN_HOURS. What happens to this attribute?

As a general rule, any attributes that are not dependent in a partial dependency will remain in the original table. Hence, we will have the following table schema:

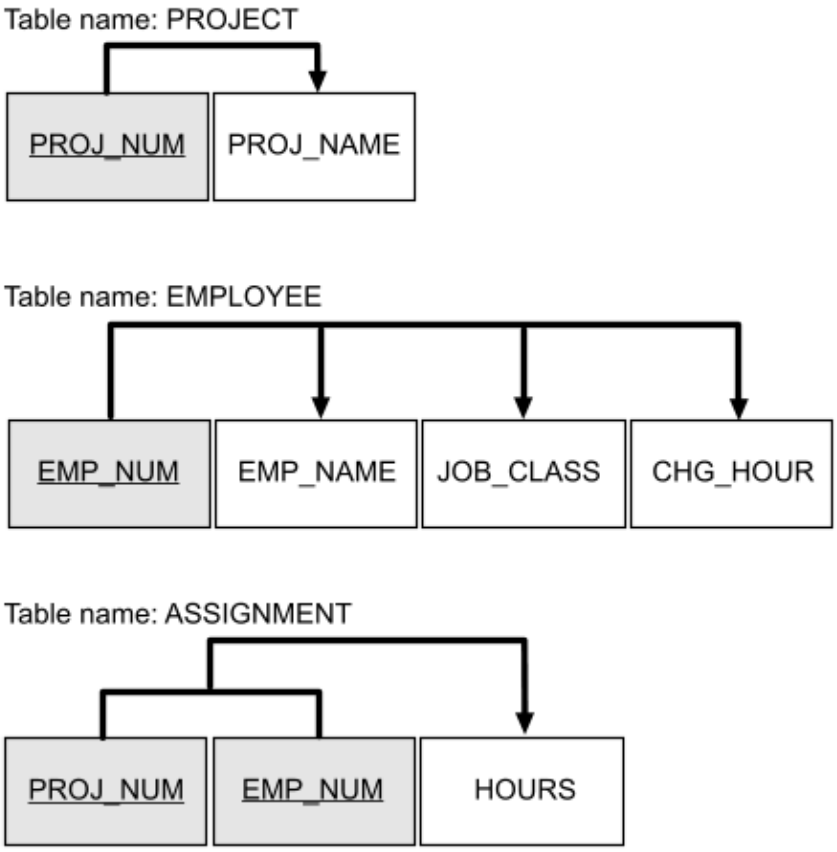
ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

As you can see, we've renamed the table to make it more relevant to its purpose, which is to record which employee is assigned to which project, and how many of his or her work hours are allocated to the specific project.

Note the following points about the ASSIGNMENT table:

1. The ASSIGNMENT table contains a composite primary key composed of the attributes PROJ_NUM and EMP_NUM.
2. By leaving the determinants in the original table as well as making them the primary keys of the new tables, we have created primary key/foreign key relationships. So, in the EMPLOYEE table, EMP_NUM is the primary key, but in the ASSIGNMENT table it is a part of the composite primary key and is a foreign key that relates the EMLOYEE table to the ASSIGNMENT table.

The new dependency diagram will look like this:



At this stage the table is in 2NF. It is now time to progress to 3NF

Conversion to third normal form

A table is in **third normal form** (**3NF**) when:

- It is in 2NF.
- It contains no transitive dependencies.

Based on the above, the task at this stage is to remove the transitive dependencies since the first condition is already fulfilled – the tables are already in 2NF.

The procedure to remove the transitive dependencies is exactly the same as that for removing partial dependencies. It is very important, to remember that the order in which this is done is critical: first the partial dependencies must be eliminated to bring the table to 2NF, and then the transitive dependencies must be eliminated.

As with 2NF, converting a table to 3NF is a two-step process.

Step 1: Make new tables to eliminate transitive dependencies

For every transitive dependency, write a copy of its determinant as a primary key for a new table. Thus, if there are three transitive dependencies, you will need to create three new tables, and each table's primary key will be the determinant in the transitive dependency.

Remember that as with the conversion to 2NF, it is important that the determinant remain in the original table to serve as a foreign key.

The PROJECTS table has only one transitive dependency, so only one new table will be created, ad its primary key will be:

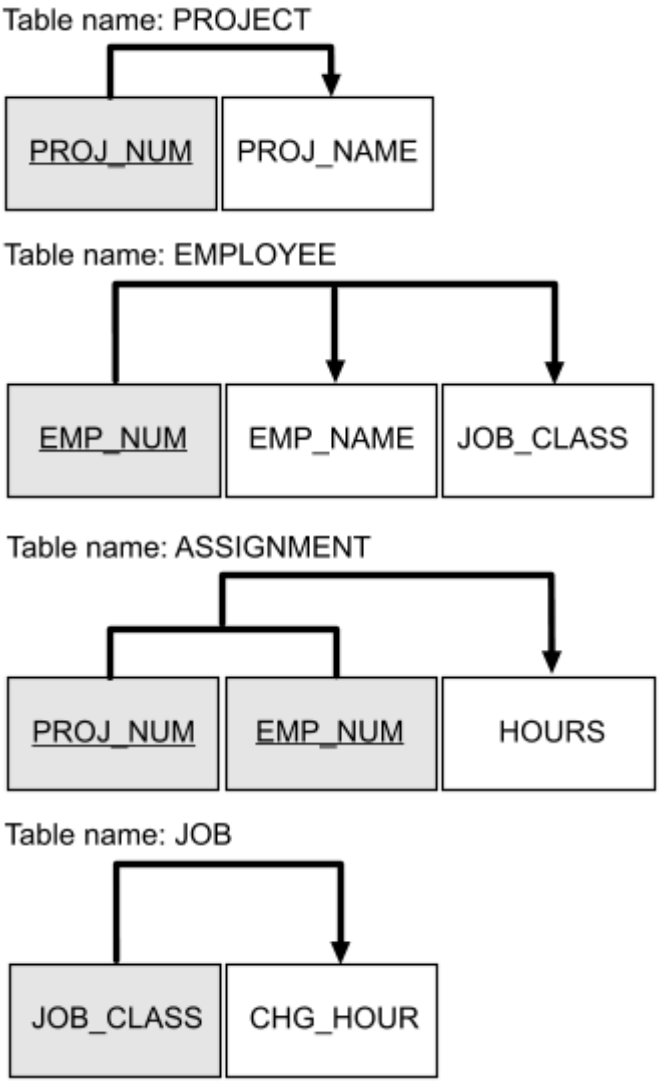
JOB_CLASS

Step 2: Reassign corresponding dependent attributes

At this stage you will use the original dependency diagram to see which attributes depend on the determinant from the previous step, and assign those attributes to the new table. The dependent attribute is then removed from the original table.

The JOB_CLASS attribute in the EMPLOYEE table has only one dependent, which is the CHG_HOUR attribute. This attribute will be moved to the new table, named JOB, as shown in the following relational schema:
JOB(**JOB_CLASS**, CHG_HOUR)

The new dependency diagram will look like this:



The database tables are now in third normal form.

Higher normal forms

Thus far we've discussed the first, second and third normal forms. These are, however, not the only normal forms; beyond the third normal form there are the following normal forms:

- The Boyce-Codd normal form (BCNF)
- The fourth normal form (4NF)
- The fifth normal form (5NF)
- The domain-key normal form (DKNF)

Bear in mind, though, that the work of the data modeller is to ensure that every table is at least in third normal form. The higher normal forms are for specialized uses or for theoretical purposes, and do not fall under the scope of most business operations.

Prescribed and Recommended Textbooks/Readings

Higher level normal forms

Read section 7.6 of Rob et al.

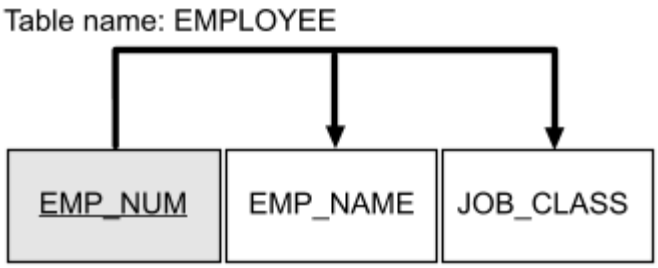
5.4 Adding further improvements to table designs

Normalization vastly improves the design of a database table, but it does not necessarily produce tables which are optimal. There may still be design issues which need addressing, but which do not fall under the scope of normalization.

Besides normalization, there are other measures which you can put into place to further improve the design of your tables. We will discuss some of these in this section.

Evaluate PK assignments

Let's take another look at the dependency diagram for the EMPLOYEE table:



Whenever a new employee is entered, a job class is also entered for that employee. The job class attribute hold values such as:

Elect. Engineer
Database Designer
Programmer
Systems Analyst
Applications Designer

Unfortunately, typing in the job classes as above lends itself to data entry errors and inconsistencies. For example, a data capture clerk can type in "Electrical Engineer" or "Elect Engineer" (without the dot) for an employee's job class instead of "Elect. Engineer". This will create a referential integrity problem. In other words, if you need to

query the database to bring back a list of all employees who are Electrical Engineers, and you run a query on the words “Elect. Engineer”, all employees whose job class is entered as “Electrical Engineer” or “Elect Engineer” or any other variation, will not come up in that list.

The solution to this problem is to introduce a JOB_CODE attribute to the JOB table. The JOB_CODE attribute will serve as a primary key in the JOB table and as a foreign key in the EMPLOYEE. The JOB_CODE attribute will ideally be a unique numeric or alpha-numeric code for each JOB_CLASS.

Evaluate naming conventions

Standard database naming conventions suggest that the table name is appended to the beginning of each attribute name. In this case the CHG_HOUR attribute will be renamed to JOB_CHG_HOUR.

Also, attribute names need to be descriptive of the contents of the attrbutes. In this case, the attribute JOB_CLASS should be renamed to JOB_DESCRIPTION to highlight the fact that employee job descriptions will be stored in it.

Refine the atomicity of the attributes

Database table attributes need to be atomic – that is, they need to store data values which cannot be subdivided into smaller parts.

As an example, consider the PROJ_NAME attribute in the PROJECT table. This attribute stores the name of a project such as “Evergreen”. Clearly the project name cannot be further subdivided into smaller parts.

In contrast to the PROJ_NAME attribute is the EMP_NAME attribute, which stores the employee’s first name, initials and surname, such as “Alice K. Johnson”. This single attribute comprises of three subdivisions: name, initials and surname. Each of these attributes needs to be a separate attribute in order to attain attribute atomicity, producing the following attributes:
EMP_FIRSTNAME, EMP_INITIALS, EMP_SURNAME

Identify New Attributes

The original PROJECTS table is a highly simplified one which we used for the purpose of our discussion. However, in a real-world situation, a typical table will have many more attributes, such as project start date, project end date, etc.

As a data modeller you will need to ensure that all possible attributes relating to entities in a table have been included.

Prescribed and Recommended Textbooks/Readings

Denormalization

Read section 7.8 of Rob et al.

5.5 Summary

- Normalization is a technique used to design tables in which data redundancies are minimized.
- The first three normal forms (1NF, 2NF, and 3NF) are most commonly encountered. From a structural point of view, higher normal forms are better than lower normal forms, because higher normal forms yield relatively fewer data redundancies in the database.
- Almost all business designs use 3NF as the ideal normal form. A special, more restricted 3NF known as Boyce-Codd normal form, or BCNF, is also used.
- A table is in 1NF when all key attributes are defined and when all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies.
- A partial dependency is one in which an attribute is functionally dependent on only a part of a multiattribute primary key.
- A transitive dependency is one in which one attribute is functionally dependent on another nonkey attribute.
- A table with a single-attribute primary key cannot exhibit partial dependencies.
- A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute. A table in 2NF may still contain transitive dependencies.
- A table is in 3NF when it is in 2NF and contains no transitive dependencies.
- A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements.
- Normalization is an important part - but only a part - of the design process. As entities and attributes are defined during the ER modelling process, the following need to be applied:
    - Evaluate PK assignments
    - Evaluate naming conventions
    - Refine the atomicity of the attributes
    - Identify New Attributes

Revision Questions

1. What is normalization?
2. What is the objective of normalization? What characteristics must each table have after the normalization process is complete?
3. Define the following terms:
    a. Superkey
    b. Candidate key
    c. Functional dependence
    d. Partial dependency
    e. Transitive dependency
4. What criteria have to be fulfilled for a table to be in:
    a. 1NF
    b. 2NF
    c. 3NF

5.6 Answers

1. Normalization is described as a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.
2. The objective of normalization is to ensure that each table in the database conforms to the concept of well-formed relations - that is, each table must have the following characteristics:
    1. Each table represents a single subject. For example, a course table will contain only data that directly pertain to courses. Similarly, a student table will contain only student data.
    2. No data item will be *unnecessarily* stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that any data which needs to be updated, is updated in only one place.
    3. All nonprime attributes in a table are dependent on the primary key - the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data are uniquely identifiable by a primary key value.
3. Definitions follow:
    1. *Superkey*: A superkey is an attribute or set of attributes that uniquely identifies rows within a table; in other words, each row is always guaranteed to have a unique superkey.
    2. *Candidate key*: a candidate key is a minimal (irreducible) superkey.
    3. *Functional dependence*: The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B.*

        Example: PROJ_NUM → PROJ_NAME

(read as *"*PROJ_NUM functionally determines PROJ_NAME*"*)

In this case, the attribute PROJ_NUM is known as the "**determinant***"* attribute, and the attribute PROJ_NAME is known as the *"***dependent***"* attribute.

4. *Partial dependency*: A partial dependency exists when there is a functional dependence in which the determinant is only part of the primary key (remember we are assuming there is only one candidate key). For example:

   If (A, B) → (C,D) and B → C and (A, B) is the primary key, then the functional dependence B → C is a partial dependency because only part of the primary key (B) is needed to determine the value of C. Partial dependencies tend to be rather straightforward and easy to identify.

5. *Transitive dependency*: A transitive dependency exists when there are functional dependencies such that:

$$X \rightarrow Y \text{ and } Y \rightarrow Z$$

And X is the primary key. In that case, the dependency X → Z is a transitive dependency because X determines the value of Z via Y. Unlike partial dependencies, transitive dependencies are more difficult to identify.

a) A table is in the first normal form (1NF) when:

1. All of the key attributes are defined.
2. There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
3. All attributes are dependent on the primary key.

b) A table is in **second normal form** (**2NF**) when:

1. It is in 1NF.
2. It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key. Note that it is still possible for a table in 2NF to exhibit transitive dependency; that is, the primary key may rely on one or more nonprime attributes to functionally determine other nonprime attributes, as is indicated by a functional dependence among the nonprime attributes.

c) A table is in **third normal form** (**3NF**) when:

1. It is in 2NF.
2. It contains no transitive dependencies.

Unit 6:

Introduction to Structure Query Language (SQL)

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
6.1. Introduction to SQL	• Know the basic commands and functions of SQL
6.2. Data Definition Commands	• Know how to use SQL for data administration (to create tables, indexes, and views) • Be able to use SQL for data manipulation (to add, modify, delete, and retrieve data) • Know how to use SQL to query a database for useful information
6.3. Summary	• Summarise topic areas covered in unit

Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

6.1 Introduction to SQL

A database language allows you to create database and table structures, to perform basic data management chores (add, delete, and modify), and to perform complex queries designed to transform the raw data into useful information. A database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. It must be portable; that is, it must conform to some basic standard so that an individual does not have to relearn the basics when moving from one RDBMS to another. SQL meets those database language requirements well.

There are two broad categories of SQL functions:

- Data definition language (DDL), and
- Data manipulation language (DML)

In data definition language, SQL includes commands to create database objects such as tables, indexes, and views, as well as commands to define access rights to those database objects. Some of the data definition commands are:

- CREATE SCHEMA AUTHORIZATION – creates a database schema
- CREATE TABLE – creates a new table in the user's database schema
- NOT NULL – ensures that a column will not have null values
- UNIQUE – ensures that a column will not have duplicate values
- PRIMARY KEY – defines a primary key for a table
- FOREIGN KEY – defines a foreign key for a table
- DEFAULT – defines a default value for a column (when no value is given)
- CHECK – validates data in an attribute
- CREATE INDEX – creates an index for a table
- CREATE VIEW – creates a dynamic subset of rows/columns from one or more tables
- ALTER TABLE – modifies a tables definition (adds, modifies, or deletes attributes or constraints)
- CREATE TABLE AS – creates a new table based on a query in the user's database schema
- DROP TABLE – permanently deletes a table (and its data)
- DROP INDEX – permanently deletes an index
- DROP VIEW – permanently deletes a view

Many of the above will be introduced later in this chapter.

Prescribed and Recommended Textbooks/Readings

Data definition commands

Study section 8.2 of Rob et al.

In data manipulation language, SQL includes commands to insert, update, delete, and retrieve data within the database tables. Some of the data manipulation commands you may come across are:

- INSERT – inserts row(s) into a table
- SELECT – selects attributes from rows in one or more tables or views
    - WHERE – restricts the selection of rows based on a conditional expression
    - GROUP BY – groups the selected rows based on one or more attributes
    - HAVING – restricts the selection of grouped rows based on a condition
    - ORDER BY – orders the selected rows based on one or more attributes
- UPDATE – modifies an attribute's values in one or more table's rows
- DELETE – deletes one or more rows from a table
- COMMIT – permanently saves data changes
- ROLLBACK – restores data to their original values
- =, <, >, <=, >=, <> - comparison operators used in conditional expressions
- AND/OR/NOT – logical operators used in conditional expressions
- BETWEEN – special operator used to check whether an attribute value is within a range
- IS NULL – special operator used to check whether an attribute value is null
- LIKE – special operator used to check whether an attribute value matches a given string pattern
- IN – special operator used to check whether an attribute value matches any value within a value list
- EXISTS – special operator used to check whether a subquery returns any rows
- DISTINCT – special operator used to limit values to unique values
- COUNT – aggregate function that returns the number of rows with non-null values for a given column
- MIN – aggregate function that returns the minimum attribute value found in a given column
- MAX – aggregate function that returns the maximum attribute value found in a given column
- SUM – aggregate function that returns the sum of all values for a given column
- AVG – aggregate function that returns the average of all values for a given column

As with the DDL functions, you will be introduced to many of these functions later in this chapter.

These AGGREGATE FUNCTIONS are used with SELECT to return any of those mathematical summaries on columns.

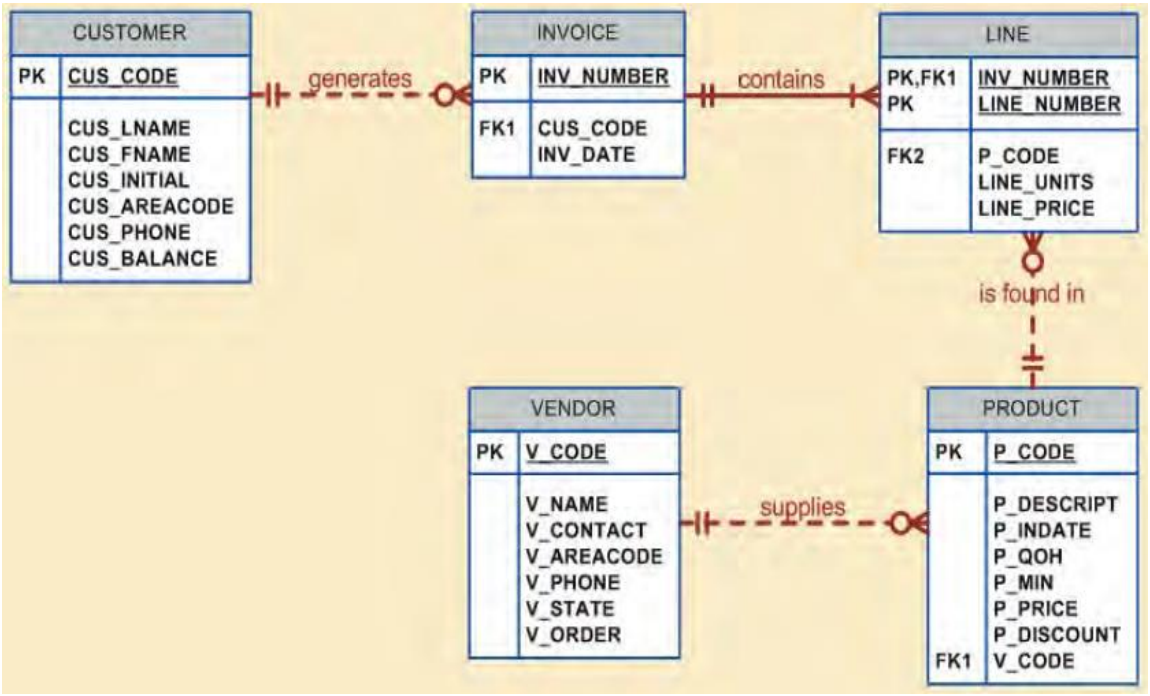
SQL is a non-procedural language that you command *what* is to be done and not *how* it is to be done. ANSI (American National Standards Institute) prescribes a standard SQL, SQL 2003, which is a currently fully approved version. The different SQL applications are Oracle, Microsoft SQL Server, MySQL, and IBM's DB2.

6.2 Data Definition Commands

The Database Model

A simple database composed of the CUSTOMER, INVOICE, LINE, PRODUCT and VENDOR tables will be used to illustrate the SQL commands in this chapter.

The tables are shown below:



This database model reflects the following business rules:

- A customer may generate many invoices. Each invoice is generated by only one customer.
- An invoice contains one or more invoice lines. Each invoice line is associated withno more than one invoice.
- Each invoice line references one product. A product may be found in many invoice lines. (You can sell more than one hammer to more than one customer.)
- A vendor *may* supply many products. Some vendors do not (yet?) supply products. (For example, a vendor list may include *potential* vendors.)
- If a product is vendor-supplied, that product is supplied by only a single vendor.
- Some products are not supplied by a vendor. (For example, some products may be produced in-house or bought on the open market.)

To have a point of reference for understanding the effect of the SQL queries, the contents of the PRODUCE and VENDOR tables (these two tables will be used to illustrate the initial set of data definition commands) are listed below.

Table: VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson. Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo Inc.	Flushing	904	215-8995	FL	N
21231	DRLE Supply	Singh	615	228-3245	JTN	
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damai Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon stems	Orlon	904	456-0092	FL	Y

Table: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09	8	5	10999	000	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	1499	0.05	21344
14-QI/U	9.00-in. pwr. saw blade	13-Nov-09	18	12	1749	0.00	21344
1546-002	Hrd. cloth, 1/4-in., 2x50	15-jan-10	15	8	39.95	0.00	23119

1558-owl	Hrd. cloth, 112-in., 3x50	15-jan-10	23	5	43.99	0.00	231 19
2232/OTY	B&D jigsaw. 12-in, blade	30-Dec-09	8	5	109.92	0.05	242 88
2232/OWE	B&D jigsaw, 8-in, blade	24-Dec-09	6	5	99.87	0.05	242 88
2238/OPD	O&D cordless drill, 1/2-in.	20-jan-10	12	5	38.95	0.05	212 25
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	1440	0.05	213 44
54778-2T	Rat-tail file. 1/8in, fine	15-Dec-09	43	20	499	0.00	242 88
89-WRE-Q	Hicut chain saw, 16 in,	07-Feb-10	11	5	256.99	0.05	
PVC23DRT	PVC pipe, 3.5-in., 5-ft	20-Feb-10	188	75	5.87	0.00	212 25
SM-i 8277	1.25-in, metal screw, 25	01-Mar-10	172	75	6.99	0.00	212 31
SW-231 16	2.5i'. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	255 95
1WR3Irr3	Steel matting 4'x8'x1/16"	17-jan-10	18	5	119.95	0.10	

Note the following about these two tables:

- The VENDOR table contains vendors who

The Database Schema

In the SQL environment, a **schema** is a group of database objects – such as tables and indexes – that are related to each other. Usually, the schema belongs to a single user or application. A single database can hold multiple schemas belonging to different users or applications. Schemas are useful in the grouping of tables by owner (or function) and enforcing security by allowing each user to see only the tables that belong to that user.

Data Types

Prescribed and Recommended Textbooks/Readings

SQL data types

Study section 8.2.4 of Rob et al, ensuring that you have a good understanding of SQL data types.

In a relational database it is essential to store various types of data in appropriate data storage types in order to facilitate speedy and efficient data storage and retrieval. Numbers, for example, need to be stored in numeric data types in order to facilitate mathematical calculations etc. Take a look at the following table:

Table Name	Attribute Name	Contents	Type
PRODUCT	P_CODE	Product code	CHAR(10)
	P_DESCRIPT	Product description	VARCHAR(35)
	P_INDATE	Stocking date	DATE
	P_QOH	Units available	SMALLINT
	P_MIN	Minimum units	SMALLINT
	P_PRICE	Price	NUMBER(8,2)
	P_DISCOUNT	Discount rate	NUMBER(5,2)
	V_CODE	Vendor code	INTEGER
VENDOR	V_CODE	Vendor code	INTEGER
	V_NAME	Vendor Name	CHAR(35)
	V_CONTACT	Contact person	CHAR(25)
	V_AREACODE	Area code	CHAR(3)
	V_PHONE	Telephone number	CHAR(8)
	V_STATE	State	CHAR(2)
	V_ORDER	Previous order	CHAR(1)

The above table shows the data dictionary for the PRODUCT and VENDOR tables, just to give you a general idea of the types of data we will be dealing with and how these data types are declared. Note the following about the data dictionary:

- P_PRICE clearly requires some kind of numeric data type, defining it as a character field is not acceptable.
- Just as clearly, a vendor name is an obvious candidate for a character data type. For example, VARCHAR(35) fits well because vendor names are "variable-length" character strings, and in this case, such strings may be up to 35 characters long.
- At first glance, it might seem logical to select a numeric data type for V_AREACODE because it contains only digits. However, we will never need to do any calculations, such as addition or subtraction, on area codes. Therefore, selecting a character data type is more appropriate.
- Selecting P_INDATE to be a (Julian) DATE field rather than a character field is desirable because the Julian dates allow you to make simple date comparisons and to perform data arithmetic. For instance, if you have used DATE fields, you can determine how many days there are between them.

Data type selection sometimes requires professional judgement. For example, you must make a decision about the V_CODE's data type as follows:

- If you want the computer to generate new vendor codes by adding 1 to the largest recorded vendor code, you must classify V_CODE as a numeric attribute.
- If you do not want to perform mathematical procedures based on V_CODE, you should classify it as a character attribute, even though it is composed entirely of numbers. Character data are "quicker" to process in queries. Therefore, when there is no need to perform mathematical procedures on the attribute, store it as a character attribute.

More common SQL data types not mentioned in the data dictionary above:

Numeric data types:

- NUMBER(L,D) – the declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place. Examples: 12.32, -134.99.
- INTEGER – may be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
- SMALLINT – like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
- DECIMAL(L,D) – like the NUMBER specification, but the storage length is a *minimum specification*. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.

Character data types:

- CHAR(L) – fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as Smith and Katzenjammer are each stored as 25 characters.
- VARCHAR(L) or VARCHAR2(L) – variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.

Date data types:

- DATE – stores dates in the Julian date format.

In addition to the above data types mentioned, SQL supports several other data types, including TIME, TIMESTAMP, REAL, DOUBLE, FLOAT, and intervals such as INTERVAL DAY TO HOUR.

Creating Table Structures

Here, we implement the CREATE TABLE for the PRODUCT and VENDOR tables. The CREATE TABLE syntax is:

CREATE TABLE *tablename* (
column1 data type [*constraint*] [,
column2 data type [*constraint*]] [,
PRIMARY KEY (*column1* [*, column2*])] [,
FOREIGN KEY (*column1* [, *column2*]) REFERENCES *tablename*] [,
CONSTRAINT *constraint*]);

The above CREATE TABLE code can all fit in one line; however, for easy readability we do our syntax in this style, using one line per column (attribute) definition. Before we continue to creating tables for PRODUCT and VENDOR, here are a few descriptions of the syntax commands we've just used:

- *Tablename* – the name of a table
- *Column* – the name of an attribute in a table
- *Data type* – a valid data-type definition
- *Constraint* – a valid constraint definition

Using the database of Figure 7.2, we can define the tables for both the VENDOR and PRODUCT, first the VENDOR:

```
CREATE TABLE VENDOR (
        V_CODE          INTEGER         NOT NULL UNIQUE,
        V_NAME          VARCHAR(35)     NOT NULL,
        V_CONTACT       VARCHAR(15)     NOT NULL,
        V_AREACODE      CHAR(3)         NOT NULL,
        V_PHONE         CHAR(8)         NOT NULL,
        V_STATE         CHAR(2)         NOT NULL,
        V_ORDER         CHAR(1)         NOT NULL,
        PRIMARY KEY(V_CODE));
```

- Because the PRODUCT table contains a foreign key that references the VENDOR table, we create the VENDOR table first.
- Many databases recognize only the data type VARCHAR2, but Oracle accepts the VARCHAR data type and automatically converts it to VARCHAR2.
- If you use the PRIMARY KEY designation in Oracle, you do not need the NOT NULL and UNIQUE specifications as done above.

Next, we will create the PRODUCT table:

```
CREATE TABLE PRODUCT (
        P_CODE          VARCHAR(10)     NOT NULL UNIQUE,
        P_DESCRIPT      VARCHAR(35)     NOT NULL,
        P_INDATE        DATE            NOT NULL,
        P_QOH           SMALLINT        NOT NULL,
        P_MIN           SMALLINT        NOT NULL,
        P_PRICE         NUMBER(8,2)     NOT NULL,
        P_DISCOUNT      NUMBER(5,2)     NOT NULL,
        V_CODE          INTEGER,
        PRIMARY KEY (P_CODE),
        FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```

- The ON UPDATE CASCADE clause is part of the ANSI standard, but it may not be supported by your RDBMS, like Oracle. In that case, you don't need to include the ON UPDATE CASCADE clause to your table.

The NOT NULL constraint ensures that the column will never contain a null value. When it is crucial to have the data available, the NOT NULL specification will not allow the end user to leave the attribute empty (with no data entry at all).

The UNIQUE specification creates a unique index in the column. Use it to avoid having duplicated values in a column. The primary key attributes contain both a NOT NULL and a UNIQUE specification because every row needs to have a primary key, and the primary key always needs to be unique, otherwise finding the row will be impossible. Hence, those specifications enforce the entity integrity requirements.

The entire table definition is enclosed in parentheses. A comma is used to separate each table element (attributes, primary key, and foreign key) definition. If you are working with a composite primary key, all of the primary key's attributes are contained within the parentheses and are separated with commas.

For example, if a table has a primary key that consists of the two attributes INV_NUMBER and LINE_NUMBER, you would define the primary key by typing:
PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
The *order of the primary key components is important* because the indexing starts with the first-mentioned attributed, then proceeds with the next attribute, and so on.

SQL Constraints
In chapter 2 you learned about the importance of following the rules of entity integrity and referential integrity in a relational database environment. These rules can be implemented via SQL. Most SQL implementations support both integrity rules.

Entity integrity is enforced automatically when the primary key is specified in the CREATE TABLE command sequence. For example, you can create the VENDOR table structure and set the stage for the enforcement of entity integrity rules by using:

PRIMARY KEY (V_CODE)

In the PRODUCT table's CREATE TABLE sequence, note that referential integrity has been enforced by specifying in the PRODUCT table:

FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE

The foreign key constraint definition ensures that:

- You cannot delete a vendor from the VENDOR table if at least one product row references that vendor. This is the default behaviour for the treatment of foreign keys.
- The ON UPDATE CASCADE constraint ensures that if a change is made in an existing VENDOR table's V_CODE, that change is reflected automatically in any PRODUCT table V_CODE reference. That

restriction makes it impossible for a V_CODE value to exist in the PRODUCT table pointing to a non-existent VENDOR table V_CODE value. In other words, the ON UPDATE CASCADE specification ensures the preservation of referential integrity.

The PRIMARY KEY and FOREIGN KEY define the following constraints:

- The NOT NULL constraint ensures that a column does not accept nulls.
- The UNIQUE constraint ensures that all values in a column are unique.
- The DEFAULT constraint assigns a value to an attribute when a new row is added to a table. The end user may enter a value other than the default value.
- The CHECK constraint is used to validate data when an attribute value is entered. The CHECK constraint does precisely what its name suggests: it checks to see that a specified condition exists. Examples of such constraints include the following:
    - *The minimum order value must be at least 10.*
    - *The date must be after April 15, 2010.*

  If the CHECK constraint is met for the specified attribute (that is, the condition is true), the data are accepted for that attribute. If the condition is found to be false, an error message is generated and the data are not accepted.

The CREATE TABLE command lets you define constraints in two different ways:

- When you create a column definition (known as a *column constraint*). A column constraint applies to just one column.
- When you use the CONSTRAINT keyword (known as a *table constraint*). A table constraint may apply to many columns.

SQL Indexes

When declaring a primary key, the DBMS automatically creates a unique index. The ability to create indexes quickly and efficiently is important.

Should you wish to create an index on any other column besides the primary key column, you can do so using the **CREATE INDEX** command:

CREATE [UNIQUE] INDEX *indexname* ON *tablename* (*column1* [, *column2*])

For example, based on the attribute P_INDATE stored in the PRODUCT table, the following command creates an index named P_INDATEX:

CREATE INDEX P_INDATEX ON PRODUCT(P_INDATE);

Using the UNIQUE index qualifier, you can create an index that prevents you from using a value that has been used before. This is especially useful when the index attribute is a candidate key whose values must not be duplicated:

CREATE UNIQUE INDEX P_CODEX ON PRODUCT(P_CODE);

Now, when creating a unique index on the primary key attribute, this allows a user to not enter a duplicate primary key (P_CODE) value because SQL will produce an error message such as “duplicate value in index”.

Deleting a Table from the Database

A table can be deleted from the database using the **DROP TABLE** command. You can delete the PART table you just created with:

DROP TABLE PART;

A table can be DROPPED only if that table is not the “one” side of any relationship.

Data Manipulation Commands

Prescribed and Recommended Textbooks/Readings

Data Manipulation Commands

Study section 8.3 of Rob et al.

As mentioned earlier, the data manipulation commands allow us to make changes to the data within the tables. These changes can be in the form of adding data, deleting data and editing data.

Adding Table Rows

The SQL command to insert a row of data is the **INSERT** command. The INSERT command’s basic syntax looks like this:

INSERT INTO *tablename* VALUES (*value1*, *value2*, …, *value n*)

Because the PRODUCT table uses its V_CODE to reference the VENDOR table’s V_CODE, an integrity violation will occur if those VENDOR table V_CODE values don’t yet exist. Therefore, you need to enter the VENDOR rows before the PRODUCT rows. The first two data rows of the VENDOR data shown in Figure 7.2 are as follows:

```
INSERT INTO VENDOR
        VALUES(21225, ‘Bryson, Inc.’, ‘Smithson’, ‘615’, ‘223-3234’, ‘TN’, ‘Y’);
INSERT INTO VENDOR
        VALUES(21226, “Superloo, Inc’, ‘Flushing’, ‘904’, ‘215-8995’, ‘FL’, ‘N’);
```

To see the contents of the VENDOR table, you can use the following command:

```
SELECT * FROM VENDOR;
```

The PRODUCT table rows would be entered in the same fashion, using the PRODUCT data shown in the data tables previously:

```
INSERT INTO PRODUCT
   VALUES(‘11QER/31’, ‘Power painter, 15 psi., 3-nozzle’, ’03-Nov-09’, 8, 5, 109.99, 0.00, 25595);
INSERT INTO PRODUCT
   VALUES(’13-Q2/P2’, ‘7.25-in. pwr. saw blade’, ’13-Dec-09’, 32, 15, 14.99, 0.05, 21344);
```

To see the contents of the PRODUCT table, as with VENDOR, use the following command:
SELECT * FROM PRODUCT;

In the preceding data entry lines, observe that:

- The row contents are entered between parentheses. Note that the first character after VALUES is a parenthesis and that the last character in the command sequence is also a parenthesis.
- Character (string) and date values must be entered between apostrophes (').
- Numerical entries are *not* enclosed in apostrophes.
- Attribute entries are separated by commas.
- A value is required for each column in the table.

This version of the INSERT commands adds one table row at a time.

Inserting Rows with Null Attributes

What do you do if a product does not have a vendor or if you don't yet know the vendor code? In those cases, you would want to leave the vendor code null. To enter a null, use the following syntax:

```
INSERT INTO PRODUCT
        VALUES('BRT-345', 'Titanium drill bit', '18-Oct-09', 75, 10, 4.50, 0.06, NULL);
```

The NULL entry will be accepted only if the V_CODE attribute does not use the NOT NULL declaration in the CREATE TABLE statement.

Saving Table Changes

Any changes made to the table contents are not saved on disk until you close the database, close the program you are using, or use the **COMMIT** command. If the database is open and a power outage or some other interruption occurs before you issue the COMMIT command, your changes will be lost and only the original table contents will be retained. The syntax for the COMMIT command is:

COMMIT [WORK]

The COMMIT command permanently saves *all* changes – such as rows added, attributes modified, and rows deleted – made to any table in the database. A simple COMMIT command is all that's needed to save your changes to any table.

However, the COMMIT command's purpose is not just to save changes. The ultimate purpose of the COMMIT and its reverse, the ROLLBACK command (discussed later in this chapter), is to ensure database update integrity in transaction management.

Listing Table Rows

The **SELECT** command is used to list the contents of a table. The syntax of the SELECT command is as follows:

SELECT *columnlist* FROM *tablename*

The *columnlist* represents one or more attributes or columns, separated by commas. An * (asterisk) can be used as a wildcard character to list all attributes. For example, to list all attributes and all rows of the PRODUCT table, use:

SELECT * FROM PRODUCT;

The output generated by that command is shown below:

Table: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09	8	5	10999	000	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	1499	0.05	21344
14-Ql/U	9.00-in. pwr. saw blade	13-Nov-09	18	12	1749	0.00	21344
1546-002	Hrd. cloth, 1/4-in., 2x50	15-jan-10	15	8	39.95	0.00	23119
1558-owl	Hrd. cloth, 112-in., 3x50	15-jan-10	23	5	43.99	0.00	23119
2232/OTY	B&D jigsaw. 12-in, blade	30-Dec-09	8	5	109.92	0.05	24288
2232/OWE	B&D jigsaw, 8-in, blade	24-Dec-09	6	5	99.87	0.05	24288
2238/OPD	O&D cordless drill, 1/2-in.	20-jan-10	12	5	38.95	0.05	21225
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	1440	0.05	21344
54778-2T	Rat-tail file. 1/8in, fine	15-Dec-09	43	20	499	0.00	24288
89-WRE-Q	Hicut chain saw, 16 in,	07-Feb-10	11	5	256.99	0.05	
PVC23DRT	PVC pipe, 3.5-in., 5-ft	20-Feb-10	188	75	5.87	0.00	21225
SM-i 8277	1.25-in, metal screw, 25	01-Mar-10	172	75	6.99	0.00	21231
SW-23116	2.5i'. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	25595
1WR3Irr3	Steel matting 4'x8'x1/16"	17-jan-10	18	5	119.95	0.10	

To show only some of the columns, let say the first three columns of the PRODUCT table, you do so like this:

SELECT P_CODE, P_DESCRIPT, P_INDATE FROM PRODUCT;

This will display only the P_CODE, P_DESCRIPT and P_INDATE columns in the table, as shown below:

Table: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09
14-Ql/U	9.00-in. pwr. saw blade	13-Nov-09
1546-002	Hrd. cloth, 1/4-in., 2x50	15-jan-10
1558-owl	Hrd. cloth, 112-in., 3x50	15-jan-10
2232/OTY	B&D jigsaw. 12-in, blade	30-Dec-09
2232/OWE	B&D jigsaw, 8-in, blade	24-Dec-09
2238/OPD	O&D cordless drill, 1/2-in.	20-jan-10
23109-HB	Claw hammer	20-Jan-10
23114-AA	Sledge hammer, 12 lb.	02-Jan-10
54778-2T	Rat-tail file. 1/8in, fine	15-Dec-09
89-WRE- Q	Hicut chain saw, 16 in,	07-Feb-10
PVC23DRT	PVC pipe, 3.5-in., 5-ft	20-Feb-10
SM-i 8277	1.25-in, metal screw, 25	01-Mar-10
SW-231 16	2.5i'. wd. screw, 50	24-Feb-10
1WR3Irr3	Steel matting 4'x8'x1/16"	17-jan-10

Updating Table Rows

Use the **UPDATE** command to modify data in a table. The syntax for this command is:

UPDATE *tablename*

SET *columnname* = *expression* [, *columnname* = *expression*]

[WHERE *conditionlist*];

For example, if you want to change P_INDATE from December 13, 2009, to January 18, 2010, in the second row of the PRODUCT table of Figure 7.3, use the primary key (13-Q2/P2) to locate the correct (second) row.
Therefore, type:

```
UPDATE PRODUCT
SET P_INDATE = '18-JAN-2010'
WHERE P_CODE = '13-Q2/P2';
```

If more than one attribute is to be updated in the row, separate the corrections with commas:

```
UPDATE PRODUCT
SET P_INDATE = '18-JAN-2010', P_PRICE = 17.99, P_MIN = 10
WHERE P_CODE = '13-Q2/P2';
```

What would have happened if the previous UPDATE command had not included the WHERE condition? The P_INDATE, P_PRICE, and P_MIN values would have been changed in *all* rows of the PRODUCT table. Remember, the UPDATE command is a set-oriented operator. Therefore, if you don't specify a WHERE condition, the UPDATE command will apply the changes to *all* rows in the specified table.
You can confirm that the changes were implemented by using the SELECT command to check the PRODUCT table's listing:

```
SELECT * FROM PRODUCT;
```

Restoring Table Contents

If you have not yet used the COMMIT command to store the changes permanently in the database, you can restore the database to its previous condition with the **ROLLBACK** command. ROLLBACK undoes any changes since the last COMMIT command and brings the data back to the values that existed before the changes were made. To restore the data to their "prechange" condition, type:

```
ROLLBACK;
```

Use the SELECT statement again to see that the ROLLBACK did restore the data to their original values.
COMMIT and ROLLBACK work only with data manipulation commands that are used to add, modify, or delete table rows. For example, assume that you perform these actions:

1. CREATE a table called SALES.
2. INSERT 10 rows in the SALES table.
3. UPDATE two rows in the SALES table.
4. Execute the ROLLBACK command.

Will the SALES table be removed by the ROLLBACK command?
No, the ROLLBACK command will undo *only* the results of the INSERT and UDPATE commands. All data definition commands (CREATE TABLE) are automatically committed to the data dictionary and cannot be rolled back.

Deleting Table Rows

It is easy to delete a table row using the **DELETE** statement; the syntax is:

DELETE FROM *tablename*

[WHERE *conditionlist*];

For example, if you want to delete from the PRODUCT table the product that you added earlier whose code (P_CODE) is 'BRT-345', use:

```
DELETE FROM PRODUCT
WHERE P_CODE = 'BRT-345';
```

In that example, the primary key value lets SQL find the exact record to be deleted. However, deletions are not limited to a primary key match; any attribute may be used. For example, in your PRODUCT table, you will see that there are several products for which the P_MIN attribute is equal to 5. Use the following command to delete all rows from the PRODUCT table for which the P_MIN is equal to 5:

```
DELETE FROM PRODUCT
WHERE P_MIN = 5;
```

Check the PRODUCT table's contents again to verify that all products with P_MIN equal to 5 have been deleted. Remember, that DELETE is a set-oriented command. Keep in mind that the WHERE condition is option. Therefore, if you do not specify a WHERE condition, *all* rows from the specified table will be deleted!

Select Queries

We fine-tune the SELECT command by adding restrictions to the search criteria, coupled with appropriate search conditions, is an incredibly powerful tool that enables you to transform data into information.

Selecting Rows with Conditional Restrictions

You can select partial table contents by placing restrictions on the rows to be included in the output. This is done by using the WHERE clause to add conditional restrictions to the SELECT statement. The following syntax enables you to specify which rows to select:

SELECT *columnlist*

FROM *tablelist*

[WHERE *conditionlist*];

The SELECT statement retrieves all rows that match the specified condition(s) – also known as the *conditional criteria* – you specified in the WHERE clause. The *conditionlist* in the WHERE clause of the SELECT statement is represented by one or more conditional expressions, separated by logical operators. The WHERE clause is optional.

If no rows match the specified criteria in the WHERE clause, you see a blank screen or a message that tells you that no rows were retrieved. For example, the query:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 21344;
```

Returns the description, date, and price of products with a vendor code of 21344:
Table: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	1499	0.05	21344
14-Ql/U	9.00-in. pwr. saw blade	13-Nov-09	18	12	1749	0.00	21344
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	1440	0.05	21344

There are numerous conditional restrictions that can be placed on the selected table contents. The comparison operators, =, <, <=, >, >=, <> or !=; can be used to restrict output.
The following example uses the “not equal to” operator:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE <> 21344;
```

The output lists all of the rows for which the vendor code is *not* 21344, as shown below:
Table: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09	8	5	10999	000	25595
1546-002	Hrd. cloth, 1/4-in., 2x50	15-jan-10	15	8	39.95	0.00	23119
1558-owl	Hrd. cloth, 112-in., 3x50	15-jan-10	23	5	43.99	0.00	23119

2232/OTY	B&D jigsaw. 12-in, blade	30-Dec-09	8	5	109.92	0.05	24288
2232/OWE	B&D jigsaw, 8-in, blade	24-Dec-09	6	5	99.87	0.05	24288
2238/OPD	O&D cordless drill, 1/2-in.	20-jan-10	12	5	38.95	0.05	21225
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	
54778-2T	Rat-tail file. 1/8in, fine	15-Dec-09	43	20	499	0.00	24288
89-WRE-Q	Hicut chain saw, 16 in,	07-Feb-10	11	5	256.99	0.05	
PVC23DRT	PVC pipe, 3.5-in., 5-ft	20-Feb-10	188	75	5.87	0.00	21225
SM-i 8277	1.25-in, metal screw, 25	01-Mar-10	172	75	6.99	0.00	21231
SW-231 16	2.5i'. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	25595
1WR3Irr3	Steel matting 4'x8'x1/16"	17-jan-10	18	5	119.95	0.10	

Using Comparison Operators on Character Attributes

Comparison operators may even be used to place restrictions on character-based attributes. Therefore, the command:

```
SELECT P_CODE, P_DESCRIPT, PQOH, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_CODE < '1558-QW1';
```

Would yield a list of all rows in which the P_CODE is alphabetically less than 1558-QW1, the output generated is shown below:

Table: PRODUCT

P_CODE	P_DESCRIPT	P_QOH	P_MIN	P_PRICE
11QER/31	Power painter. 15 psi ,3-nozzle	8	5	10999
13-Q2/P2	7.25-in. pwr. saw blade	32	15	1499
14-Ql/L3	9.00-in. pwr. saw blade	18	12	1749
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15	8	39.95

Using Comparison Operators on Dates

Date procedures are often more software-specific than other SQL procedures. For example, the query to list all of the rows in which the inventory stock dates occur on or after January 20, 2010 will look like this:

```
SELECT P_DESCRIPT, P_QOH, P_MIN, P_PRICE, P_INDATE
FROM PRODUCT
WHERE P_INDATE >= '20-Jan-2010';
```

The above query will produce the following result:

Table: PRODUCT

P_DESCRIPT	P_QOH	P_MIN	P_PRICE	P_INDATE
B&D cordless drill, 1/2-in.	12	5	38.95	20-jan-10
Claw hammer	23	10	9.95	20-Jan-10
Hicut chain saw, 16 in,	11	5	256.99	07-Feb-10
PVC pipe, 3.5-in., 5-ft	188	75	5.87	20-Feb-10
1.25-in, metal screw, 25	172	75	6.99	01-Mar-10
2.5i'. wd. screw, 50	237	100	8.45	24-Feb-10

Logical Operators: AND, OR, and NOT

If you want a list of the table contents for either the V_CODE = 21344 **or** the V_CODE = 24288, you can use the **OR** operator, as in the following command sequence:

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 21344 OR V_CODE = 24288;
```

This command generates the six rows that match the logical restriction, as shown below:

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-09	1499	21344
9.00-in. pwr. saw blade	13-Nov-09	1749	21344
B&D jigsaw. 12-in, blade	30-Dec-09	109.92	24288
B&D jigsaw, 8-in, blade	24-Dec-09	99.87	24288
Sledge hammer, 12 lb.	02-Jan-10	1440	21344
Rat-tail file. 1/8in, fine	15-Dec-09	499	24288

The logical **AND** has the same SQL syntax requirement. The following command generates a list of all rows for which P_PRICE is less than %50 and for which P_INDATE is a date occurring after January 15, 2010:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE P_PRICE < 50
AND P_INDATE > '15-Jan-2010';
```

This command will produce the output shown in Figure 7.13.

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-jan-10	38.95	21225
Claw hammer	20-Jan-10	9.95	
PVC pipe, 3.5-in., 5-ft	20-Feb-10	5.87	21225
1.25-in, metal screw, 25	01-Mar-10	6.99	21231
2.5i'. wd. screw, 50	24-Feb-10	8.45	25595

You can combine the logical OR with the logical AND to place further restrictions on the output. For example, suppose that you want a table listing for the following conditions:

- The P_INDATE is after January 15, 2010, and the P_PRICE is less than $50.
- Or the V_CODE is 24288.

The required listing can be produced by using:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
```

```
WHERE (P_PRICE < 50 AND P_INDATE > '15-Jan-2010')
OR V_CODE = 24288;
```

This query yields the output shown in below:

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw. 12-in, blade	30-Dec-09	109.92	24288
B&D jigsaw, 8-in, blade	24-Dec-09	99.87	24288
B&D cordless drill, 1/2-in.	20-jan-10	38.95	21225
Claw hammer	20-Jan-10	9.95	
Hicut chain saw, 16 in,	07-Feb-10	256.99	
PVC pipe, 3.5-in., 5-ft	20-Feb-10	5.87	21225
1.25-in, metal screw, 25	01-Mar-10	6.99	21231
2.5i'. wd. screw, 50	24-Feb-10	8.45	25595

The logical operator **NOT** is used to negate the result of a conditional expression. That is, in SQL, all conditional expressions evaluate to true or false. If an expression is true, the row is selected; if an expression is false, the row is not selected.

The NOT logical operator is typically used to find the rows that *do not* match a certain condition. For example, if you want to see a listing of all rows for which the vendor code is not 21344, use the command sequence:

```
SELECT *
FROM PRODUCT
WHERE NOT (V_CODE = 21344);
```

It is highly recommended for clarity to enclose the NOT condition in parentheses, however, it is optional. The logical NOT can also be combined with AND and OR.

Additional Select Query Keywords

Ordering a Listing

The **ORDER BY** clause:

SELECT *columnlist*
FROM *tablelist*
[WHERE *conditionlist*]
[ORDER BY *columnlist* [ASC | DESC]];

The default order is ascending. If you want the contents of the PRODUCT table listed by P_PRICE in ascending order:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE;
```

Note that ORDER BY yields an ascending price listing:

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
PVC23DRT	PVC pipe, 3.5-in., 5-ft	20-Feb-10	5.87
SM-i 8277	1.25-in, metal screw, 25	01-Mar-10	6.99
SW-231 16	2.5i'. wd. screw, 50	24-Feb-10	8.45
23109-HB	Claw hammer	20-Jan-10	9.95
2238/OPD	O&D cordless drill, 1/2-in.	20-Jan-10	38.95
1546-002	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	39.95
1558-owl	Hrd. cloth, 112-in., 3x50	15-Jan-10	43.99
2232/OWE	B&D jigsaw, 8-in, blade	24-Dec-09	99.87
2232/OTY	B&D jigsaw. 12-in, blade	30-Dec-09	109.92
1WR3Irr3	Steel matting 4'x8'x1/16"	17-Jan-10	119.95
89-WRE- Q	Hicut chain saw, 16 in,	07-Feb-10	256.99
54778-2T	Rat-tail file. 1/8in, fine	15-Dec-09	499.00
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	1440.00
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	1499.00
14-QI/U	9.00-in. pwr. saw blade	13-Nov-09	1749.00
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09	10999.00

Although ORDER BY produces a sorted output, the actual table contents are unaffected by the ORDER BY command. To produce the list in descending order:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE DESC;
```

Listing Unique Values

The **DISTINCT** clause produces a list of only those values that are different from one another.

```
SELECT DISTINCT V_CODE FROM PRODUCT;
```

The code yields only the different (distinct) vendor codes (V_CODE) that are encountered in the

PRODUCT table, as shown below:

V_CODE
21255
21231
21344
23119
24288
25595

Aggregate Functions

- COUNT – the number of rows containing non-null values
- MIN – the minimum attribute value encountered in a given column
- MAX – the maximum attribute value encountered in a given column
- SUM – the sum of all values for a given column
- AVG – the arithmetic mean (average) for a specified column

COUNT

The **COUNT** function is used to tally the number of non-null values of an attribute. COUNT can be used in conjunction with the DISTINCT clause.

The COUNT aggregate function uses one parameter within parentheses, generally a column name such as COUNT (V_CODE) or COUNT (P_CODE). The parameter may also be an expression such as COUNT (DISTINCT V_CODE) or COUNT (P_PRICE + 10). Using that syntax, COUNT always returns the number of non-null values in the given column. However, the syntax COUNT (*) returns the number of total rows returned by the query, including the rows that contain nulls.

6.3 Summary

- Relational databases use the SQL language to reate database and table structures, to perform basic data management chores (add, delete, and modify), and to perform complex queries designed to transform the raw data into useful information.

- SQL is portable; that is, it can be easily moved from one RDBMS to another.

- There are two broad categories of SQL functions:
  - Data definition language (DDL), and
  - Data manipulation language (DML)

- The DDL functions include, among other functions, the following:
  - CREATE SCHEMA AUTHORIZATION – creates a database schema
  - CREATE TABLE – creates a new table in the user's database schema
  - NOT NULL – ensures that a column will not have null values
  - UNIQUE – ensures that a column will not have duplicate values
  - PRIMARY KEY – defines a primary key for a table
  - FOREIGN KEY – defines a foreign key for a table
  - DEFAULT – defines a default value for a column (when no value is given)
  - CHECK – validates data in an attribute
  - CREATE INDEX – creates an index for a table
  - CREATE VIEW – creates a dynamic subset of rows/columns from one or more tables
  - ALTER TABLE – modifies a tables definition (adds, modifies, or deletes attributes or constraints)
  - CREATE TABLE AS – creates a new table based on a query in the user's database schema
  - DROP TABLE – permanently deletes a table (and its data)
  - DROP INDEX – permanently deletes an index
  - DROP VIEW – permanently deletes a view

- The DML functions include, among other functions, the following:
  - INSERT – inserts row(s) into a table
  - SELECT – selects attributes from rows in one or more tables or views
    - WHERE – restricts the selection of rows based on a conditional expression
    - GROUP BY – groups the selected rows based on one or more attributes
    - HAVING – restricts the selection of grouped rows based on a condition
    - ORDER BY – orders the selected rows based on one or more attributes
  - UPDATE – modifies an attribute's values in one or more table's rows
  - DELETE – deletes one or more rows from a table
  - COMMIT – permanently saves data changes

- ROLLBACK – restores data to their original values
- =, <, >, <=, >=, <> - comparison operators used in conditional expressions
- AND/OR/NOT – logical operators used in conditional expressions
- BETWEEN – special operator used to check whether an attribute value is within a range
- IS NULL – special operator used to check whether an attribute value is null
- LIKE – special operator used to check whether an attribute value matches a given string pattern
- IN – special operator used to check whether an attribute value matches any value within a value list
- EXISTS – special operator used to check whether a subquery returns any rows
- DISTINCT – special operator used to limit values to unique values
- COUNT – aggregate function that returns the number of rows with non-null values for a given column
- MIN – aggregate function that returns the minimum attribute value found in a given column
- MAX – aggregate function that returns the maximum attribute value found in a given column
- SUM – aggregate function that returns the sum of all values for a given column
- AVG – aggregate function that returns the average of all values for a given column

Revision Questions

1. List the criteria for a good database language
2. Differentiate between a DDL and a DML
3. What is authentication in the context of enterprise databases? Why is it important?
4. Assuming you had a table called STUDENT in a database. What command would you use to delete the entire table?

5. What will the following line of SQL do:
   INSERT INTO STUDENT VALUES(21225, 'Smith', 'John', '011 456 7890', 'Johannesburg');
6. What command is used to list table rows?
7. What does the UPDATE command do?
8. State the functions of the COMMIT and ROLLBACK commands.
9. State what the following statement will do:
   SELECT * FROM cars WHERE MANUFACTURER="Toyota";
   a.

6.4 Answers

1. A database language should have the following characteristics:
    a) A database language should allow you to create database and table structures
    b) It should allow you to perform basic data management chores (add, delete, and modify), and to perform complex queries designed to transform the raw data into useful information.
    c) A database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn.
    d) It must be portable; that is, it must conform to some basic standard so that an individual does not have to relearn the basics when moving from one RDBMS to another.

2. A data definition language (DDL), is a set of instructions used to define data structures within a database. Data manipulation language (DML) is a set of instructions used to perform operations on data in a database, such as adding and removing rows, sorting and searching.

3. Authentication is the process through which the DBMS verifies that only registered users may access the database. In an enterprise RDBMS, every user ID is associated with a database schema. Authentication is important for 2 reasons:
    a) To keep out unauthorized people
    b) To keep a record of who is making changes to the database.

4. DROP TABLE STUDENT;

5. The SELECT command

6. It modifies an existing record in a table.

7. COMMIT: The COMMIT command permanently saves *all* changes – such as rows added, attributes modified, and rows deleted – made to any table in the database. ROLLBACK: ROLLBACK undoes any changes since the last COMMIT command and brings the data back to the values that existed before the changes were made.

8. It will bring back all attributes of all cars whose manufactures attribute is “Toyota” from a table called “cars”.

Unit 7:

Transaction Management and Concurrency Control

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
7.1 Transaction Management and Concurrency Control: Introduction	• Know about database transactions and their properties
7.2 What is a Transaction?	• Understand what concurrency control is and what role it plays in maintaining the database'sintegrity
7.3 Transaction Properties	• Understand what locking methods are and how they work
7.4 Concurrency Control	• Explain how stamping methods are used for concurrency control
7.5 Concurrency Control with Locking Methods	• Explain how optimistic methods are used for concurrency control
7.6 Lock Granularity	• Describe how database recovery management is used to maintain database integrity • Explain database recovery management
7.7 Database Recovery Management	
7.8 Summary	• Summarise topic areas covered in unit

Prescribed and Recommended Textbooks/Readings

Prescribed Textbook:

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

7.1 Transaction Management and Concurrency Control: Introduction

In this chapter you will learn about two very important concepts relating to databases: transaction management and concurrency control.

The data warehouse revolution

The Business Vignette on page 593 of the prescribed book discusses the phenomenal rate of growth of databases globally in the past 20 years or so. Reading it will provide you with perspective on the present state of database technology, as well as future trends.

Read the Business Vignette on page 593 of the prescribed book.

Database transactions reflect real-world transactions that are caused by events, such as buying a product, registering for a course, or making a deposit into bank account. A transaction usually involves a number of steps, all of which must be completed successfully to prevent data integrity problems. For example, when a student registers for a course at a college, a number of changes need to be made to the database; if any one of these changes is left out, the database will contain incomplete, possibly misleading information. For example, if the accounts system is not updated to include the student's details, he will never be billed! In order to avoid the problem of incomplete transactions, relational database management systems have built-in transaction management functions.

Some of the main transaction properties are atomicity, consistency, isolation and durability. You will become familiar with these in this chapter.

The term "concurrent transactions" refers to the scenario where many transactions take place at the same time. For example, in a web database belonging to an ecommerce site, there are possibly dozens of transactions taking place every second. Similarly, a bank's database is typically accessed and modified many times each second. Concurrent transactions like these, unless properly managed, can lead to serious problems, as you will see later in this chapter. In order to prevent such issues, relational database systems need to have mechanisms in place to manage concurrent transactions. **Concurrency control** refers to managing the execution of concurrent transactions. It is most important in a multiuser database environment.

7.2 What is a Transaction?

Prescribed and Recommended Textbooks/Readings

What is a transaction?

Study section 12.1 of the prescribed book. Make sure you understand the concept of database transactions thoroughly.

Consider the student registration scenario mentioned above. When a student registers at a college, the following steps need to be carried out by the database management system, among others:

- The student's personal and contact details must be stored in the STUDENT database table
- The FEES table needs to be updated with the student's fee amount, based on the courses he chooses
- The student's name need to be included in the appropriate course register table for each course that he selects

Now consider possible scenarios which may arise if one or more of the above steps is omitted. If the accounts system is not updated with the student's fees, then the student will never be billed, and he will be able complete his studies without paying a cent. Similarly, if the student's name is not included in the course register, he will officially not be registered for the course, and will not be permitted to write the examinations! As you can see, it is imperative that each step in the above transaction is carried out completely in order to prevent major problems.

A transaction is any action that reads from and/or writes to a database. It may consist of a simple SELECT statement or more a complex combination of statements such as UPDATE and INSERT statements. A transaction must be entirely completed or entirely aborted – it should never remain half-complete. Intermediate states are not accepted.

If any of the SQL statements fail, the entire transaction is rolled back to the original database state that existed before the transaction started. A successful transaction changes the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied. Every transaction must begin with the database in a known consistent state. This ensures consistency of the database.

7.3 Transaction Properties

Each transaction must display atomicity, consistency, isolation and durability. These are sometimes known as the ACID test. When executing many transactions, the DBMS must schedule the concurrent execution of the transaction's operations. This schedule must exhibit serializability. These properties are described below:

- **Atomicity:** all operations (SQL requests) of a transaction must be completed. If not, the transaction is cancelled.
- **Consistency:** The database must always be consistent. A transaction takes a database from one consistent state to another consistent.

- **Isolation:** the data used by the execution of a transaction cannot be used by another transaction until the first transaction is complete.
- **Durability:** ensures that the changes made by a transaction cannot be undone or lost once they have been committed (saved), even if there is a system failure.
- **Serializability:** ensures that the schedule for the simultaneous execution of the transactions produces consistent results.

Transaction Management with SQL

Transaction support is provided by two SQL statements: COMMIT and ROLLBACK, both of which you've been introduced to previously. When a user of application program initiates a transaction sequence, the sequence must continue through all succeeding SQL statements until one of the following four events occurs:

- A COMMIT statement is reached: all changes are permanently recorded within the database.
- A ROLLBACK statement is reached: all changes are aborted (cancelled) and the database is rolled back to its previous consistent state.
- The end of the program is successfully reached: all changes are permanently recorded within the database.
- The program is abnormally terminated: the changes made in the database are aborted and the database is rolled back to its previous consistent.

The Transaction Log

A DBMS uses a **transaction log** to keep track of all transactions that update the database. The information stored in the log is used by the DBMS for a recovery which is triggered by a ROLLBACK statement, a program's abnormal termination or a system failure, such as network inconsistency or a disk crash.

The DBMS automatically updates the transaction log. The transaction log stores:

- A record for the beginning of the transaction.
- For each transaction component (SQL statement):
    - The type of operation being performed (updates, delete, insert).
    - The names of the object affected by the transaction (the name of the table).
    - The "before" and "after" values of the fields being updated.
    - Pointers to the previous and next transaction log entries for the same transaction.
- The ending (COMMIT) of the transaction.

Using a transaction log increases the processing overhead of a DBMS, and tends to slow things down a little, but it is worth using because of its ability to restore a corrupted database.

The transaction log is implemented as one or more files that are managed separately from the actual database files. As a result, the log files can be stored in a separate location from the actual database, which safeguards them against a system failure.

7.4 Concurrency Control

Concurrency on a global scale

Every day Google answers more than one billion questions from people around the globe in 181 countries and 146 languages. That amounts to just under 12000 searches every second! As you can well imagine, this places a lot of strain on the company's databases which store the information that people are searching for. At the same time, these databases are being constantly updated with new information.

How do you think Google's databases handle such heavy traffic without any issues?

Concurrency control refers to managing the simultaneous execution of transactions in a multiuser database system. It ensures the serializability of transactions in a multiuser database environment. Concurrency control is an extremely important function of an RDBMS because transactions executing at the same time can cause data integrity and consistency problems. The three main problems are lost updates, uncommitted data and inconsistent retrievals.

Lost Updates

The **lost update** problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost.

To better understand this, consider the following example:

Say a company database has a PRODUCT table which has a P_QOH attribute, which contains the quantities currently on hand of each product:

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter. 15 psi ,3-nozzle	03-Nov-09	24	5	10999	000	25595

As you can see from the above table, there are 24 items currently in stock. Now suppose that 2 separate transactions take place on this product at the same time: a purchase transaction and a sale transaction; that is,

the company purchases 20 more of the product from its supplier, and sells 17. These transactions are shown in the following table:

Transaction	Result
T1: Purchase 20 units	P_QOH = P_QOH + 20 = 44
T2: Sell 17 units	P_QOH = P_QOH – 17 = 27

The correct end result would be that the number of units in stock is 27.

Now suppose that the above transactions are not carried out in a serialized fashion; that is, they are not carried out one-after-the-other. This will occur if T2 gains access to the product before T1 commits. In such a case, the following sequence of steps will take place:

Time	Transaction	Step	Value
1	T1	Read P_QOH	20
2	T2	Read P_QOH	20
3	T1	P_QOH = 20 + 24 (purchased 24. Equates to 44)	Not committed
4	T2	P_QOH = 20 – 17 (sold 17. Equates to 3)	Not committed
5	T1	Write P_QOH (lost update)	44
6	T2	Write P_QOH (overwrites previous update)	3

Can you see the dire consequences which may arise if the transactions are not carried out in sequence? Fortunately, the RDBMS is equipped with functionality to prevent such occurrences.

Uncommitted Data

Uncommitted data occurs when two transactions, T1 and T2, are executed at the same time and the first transaction is rolled back after the second transaction has already accessed the uncommitted data.

To understand this, let's consider the same transactions which we used in the previous section under "Lost updates". As shown previously, a successful transaction with serial execution will produce the following result:

Transaction	Result
T1: Purchase 20 units	P_QOH = P_QOH + 20 = 44
T2: Sell 17 units	P_QOH = P_QOH – 17 = 27

The following table shows a step-by-step execution of the above transaction:

Time	Transaction	Step	Value
1	T1	Read P_QOH	20
2	T1	P_QOH = 20 + 24	
3	T1	Write P_QOH	44
4	T1	****ROLLBACK****	20
5	T2	Read P_QOH	20
6	T2	P_QOH = 20 -17	
7	T2	Write P_QOH	3

As we mentioned, the above scenario will prevail if the transactions are serialized; that is, carried out in sequence. Let's consider the scenario where there is no serialization:

Time	Transaction	Step	Value
1	T1	Read P_QOH	20
2	T1	P_QOH = 20 + 24	
3	T1	Write P_QOH	44
4	T2	Read P_QOH (Not yet committed!)	44
5	T2	P_QOH = 44 -17	27
6	T1	****ROLLBACK ****	20
7	T2	Write P_QOH	27

Clearly, this will read to a disastrous situation, since there are only 3 stock units available in reality, but the database shows 27.

Inconsistent Retrievals

Inconsistent retrievals occur when a transaction accesses data before and after another transaction(s) finish working with that data. For example, an inconsistent retrieval would occur if transaction T1 calculated some summary (aggregate) over a set of data while another transaction (T2) was updating the same data. The transaction may read some data before they are changed and other data after they are changed. This would produce inconsistent results.

The Scheduler

In the examples in the previous sections, the operations within the transaction were executed in an arbitrary order. There was no particular order in which the transactions were selected for execution. This is perfectly fine as long as two transactions, T1 and T2, access *unrelated* data; but if the transactions operate on related (or the same) data, conflict is possible among the transaction components and the selection of one execution order over another might have some undesirable consequences. So how is the correct order determined, and who determines that order? Fortunately, the DBMS handles that tricky assignment by using a built-in scheduler.

The **scheduler** is a special DBMS process that establishes the order in which the operations within which concurrent transactions are executed. The execution of database operations is performed in a way that ensures serializability (transactions are executed one after another) and isolation (when one transaction is busy on a data set, no other transaction is allowed access to the same data) of transactions. To determine a suitable order, the scheduler bases its actions on concurrency control algorithms, such as locking or time stamping methods.

However, not all transactions are serializable. The RDBMS determines which transactions are not serializable, and executes those on a first-come-first-served basis.

The scheduler also makes sure that the computer's central processing unit (CPU) and storage systems are used efficiently. It also facilitates data isolation. This ensures that two transactions do not update the same data element at the same time. Database operations may require READ and/or WRITE actions that produce conflicts.

Locking, time stamping, and optimistic methods are used by the scheduler to ensure the serializability of transactions. In the next section we look at locking in more detail.

7.5 Concurrency Control with Locking Methods

A lock guarantees exclusive use of a data item to a current transaction. This means that when one transaction is working on a specific data set, no other transaction will be able to access the same data. All transactions which need to use the data will have to wait in a queue until the current transaction completes successfully.

A lock works in this way: A transaction obtains a lock on a data set (this set may be a table, page, row or field, depending on the **lock granularity** – see the next section) before it can access data. This lock prevents any other transactions from accessing the data. The transaction retains this lock until the transaction is complete, and then the lock is released (unlocked) so that another transaction can lock the data item for its exclusive use.

Most multiuser DBMS's automatically initiate and enforce locking procedures. A **lock manager** manages all lock information. It is responsible for assigning and monitoring the locks used by the transactions.

7.6 Lock Granularity

Lock granularity specifies the level of lock use. Locking can take place in the following levels: database, table, page, row or field (attribute).

Database Level

In a **database-level lock**, the whole database is locked down by a single transaction. This prevents another transaction from using any tables in the database while the current transaction is being executed. This is good for batch processes, but not for multiuser DBMSs - imagine the chaos which would follow if every Facebook user had to wait in a queue while one person updates his profile! Clearly this is because it would be very slow if thousands of transactions had to wait for the previous transaction to be complete before the next one could reserve the database.

Table Level

In a **table-level lock**, the whole table is locked. This prevents access to any row in the table by a transaction while another transaction is using the table. If a transaction needs access to many tables, each table may be locked. However, two transactions can access the same database as long as they access different tables.

Table-level locks are less restrictive that database-level locks. However they cause traffic jams when many transactions are waiting to access the same table. They are not suitable for multiuser DBMSs.

Page Level

In a **page-level lock**, the DBMS locks an entire diskpage. A **diskpage**, or **page**, is the same as a *diskblock*. A **diskblock** is a section of a disk that can be accessed directly. A page has a fixed size, such as 4K, 8K or 16K.
A table can extend across many pages. A page can contain many rows of one or more tables. Page-level locks are the most frequently used multiuser DBMS locking method.

Row Level

A **row-level lock** is less restrictive that the other locks. The DBMS allows concurrent transactions to access different rows of the same table even when the rows are located on the same page. It improves the availability of data. However, it requires high overhead to manage. This is because a lock exists for each row in a table of the database involved in a conflicting transaction. Modern DBMSs automatically escalate a lock from a row-level to a page-level lock when the application requests many locks on the same page.

Field Level

The **field-level lock** is the least restrictive lock type – it allows parallel transactions to access the same row as long as they use different fields (attributes) within that row. It results in the most flexible multiuser access.

However, it is rarely implemented in a DBMS. This is because it requires a very high level of computer overhead and also because the row-level lock is more useful.

Deadlocks

A deadlock occurs when two transactions wait forever for each other to unlock data. For example, a deadlock occurs when two transactions, T1 and T2, exist in the following mode:

T1 = access data items X and Y

T2 = access data items Y and X

If T1 has not locked data item Y, T2 cannot begin; if T2 has not unlocked data item X, T1 cannot continue. Therefore, T1 and T2 wait for each other to unlock the required data item.

A deadlock is also called a **deadly embrace**.

Since many transactions can be executed at the same time, the possibility of generating deadlocks is increased. Deadlocks are only possible when one of the transactions want to obtain an exclusive lock on a data item. No deadlock condition exists among shared locks.

The three basic techniques to control deadlocks are:

1. ***Deadlock prevention:*** A transaction requesting a new lock is cancelled when there is a possibility that a deadlock may occur.
    - If the transaction is cancelled, all changes made by it are rolled back and all locks obtained by it are released.
    - The transaction is then rescheduled for execution.
    - Deadlock prevention works because it avoids the conditions that lead to deadlocking.

2. ***Deadlock detection:*** The DBMS periodically tests the database for deadlocks.
    - If a deadlock is found, one of the transactions (the "victim") is cancelled (rolled back and restarted) and the other transaction continues.
3. ***Deadlock avoidance:*** The transaction must obtain all the locks it needs before it can be executed.
    - It avoids the rolling back of conflicting transactions by demanding that locks be obtained in succession.
    - However, the serial lock assignment required here increases action response times.

Activity

Lock types

Research and write a detailed description of each of the various lock types that databases use. You will find adequate information on the topic in section 12.3.2 of the prescribed book.

7.7 Database Recovery Management

Database recovery restores a database from a given (usually inconsistent) to a previously consistent state. Recovery techniques are based on the **atomic transaction property**. This property states that all parts of the transaction must be treated as a single, logical unit of work where all operations are applied and completed to produce a consistent database. If any transaction operation cannot be completed, the transaction must be cancelled and any changes to the database must be rolled back (undone). Transaction recovery reverses all the changes that the transaction made to the database before the transaction was cancelled.

Recovery techniques also apply to the database and to the system after some type of critical error occurs. Critical events can cause a database to become non-operational and compromise the integrity of the data. Examples of critical events are:

- *Hardware/software failures:* Hardware failures include hard disk media failure, a bad capacitor on a motherboard, or a failing memory bank. Software failures include application program or operating system errors that cause data to be overwritten, deleted or lost.
- *Human-caused Incidents:* There are two types: unintentional or intentional.
    - An unintentional failure is caused by the carelessness of end users. These include deleting the wrong rows from a table, pressing the wrong key on the keyboard, or shutting down the main database server by accident.
    - Intentional events are more sever. They indicate that company data are at more serious risk. These include security threats caused by hackers trying to obtain unauthorized access to data resources as well as virus attacks caused by unhappy employees trying to interrupt the database operation and damage the company.
- *Natural disasters*: Includes fires, earthquakes, floods and power failures.

Transaction Recovery

In the previous sections you learned about the transaction log and how it contains data for database recovery purposes. Database transaction recovery uses data in the transaction log to recover a database from an inconsistent state to a consistent state.

Transaction recovery procedures generally the following techniques:

- Deferred-write
- Write-through techniques

When the recovery procedure uses a **deferred-write technique** (also called a **deferred update**), the transaction operations do not update the physical database immediately. Only the transaction log is updated. The database is only physically updated after the transaction reaches its commit point, using information from the transaction

log. If the transaction cancels before it reaches its commit point, no changes (ROLLBACK or undo) are made to the database because it was never updated.

When the recovery procedure uses a **write-through technique** (also called an **immediate update**), the database is updated immediately by transaction operations during the transaction's execution, before the transaction reaches its commit point. If the transaction is cancelled before it reaches its commit point, a ROLLBACK or undo operation must be performed. This restores the database to a consistent state. The ROLLBACK operation will use the transaction log "before" values.

7.8 Summary

- A transaction is a sequence of database operations that access the database. A transaction must be a logical unit of work; that is, no portion of the transaction can exist by itself.
- Either *all* parts are executed or the transaction is aborted.
- A transaction takes a database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.
- Transactions have four main properties:
    - Atomicity (all parts of the transaction are executed; otherwise, the transaction is aborted),
    - Consistency (the database's consistent state is maintained),
    - Isolation (data used by one transaction cannot be accessed by another transaction until the first transaction is completed), and
    - Durability (the changes made by a transaction cannot be rolled back once the transaction is committed).
- In addition to the above, transaction schedules have the property of serializability (the result of the concurrent execution of transactions is the same as that of the transactions being executed in serial order).
- SQL provides support for transactions through the use of two statements:
    - COMMIT (saves changes to disk)
    - ROLLBACK (restores the previous database state).
- The transaction log keeps track of all transactions that modify the database. The information stored in the transaction log is used for recovery (ROLLBACK) purposes.
- Concurrency control coordinates the simultaneous execution of transactions. The concurrent execution of transactions can result in three main problems: lost updates, uncommitted data, and inconsistent retrievals.
- The scheduler is responsible for establishing the order in which the concurrent transaction operations are executed. The transaction execution order is critical and ensures database integrity in multiuser database systems.

- Locking, time stamping, and optimistic methods are used by the scheduler to ensure the serializability of transactions.
- A lock guarantees unique access to a data item by a transaction. The lock prevents one transaction from using the data item while another transaction is using it. There are several levels of locks: database, table, page, row, and field level locks.
- When two or more transactions wait indefinitely for each other to release a lock, they are in a deadlock, also called a deadly embrace. There are three deadlock control techniques: prevention, detection, and avoidance.
- Database recovery restores the database from a given state to a previous consistent state. Database recovery is triggered when a critical event occurs, such as a hardware error or application error.
-

Revision Questions

1. List and describe the properties that all database transactions should display.
2. What does the DBMS use a transaction log to do?
3. What does the transaction log store?
4. What does concurrency control refer to, and why is it necessary?
5. List and briefly explain the three main problems which may arise if there is no concurrency control.
6. What is the purpose of the scheduler in a DBMS?
7. Most multiuser DBMS's automatically initiate and enforce locking procedures. Describe the term "lock granularity" and list and explain the four levels of lock granularity.

7.9 Answers

1. These properties are described below:

- **Atomicity:** all operations (SQL requests) of a transaction must be completed. If not, the transaction is cancelled.
- **Consistency:** The database must always be consistent. A transaction takes a database from one consistent state to another consistent.
- **Isolation:** the data used by the execution of a transaction cannot be used by another transaction until the first transaction is complete.
- **Durability:** ensures that the changes made by a transaction cannot be undone or lost once they have been committed (saved), even if there is a system failure.
- **Serializability:** ensures that the schedule for the simultaneous execution of the transactions produces consistent results.

2. A DBMS uses a **transaction log** to keep track of all transactions that update the database.
3. The transaction log stores the following items:

- A record for the beginning of the transaction.
- For each transaction component (SQL statement):
  - The type of operation being performed (updates, delete, insert).
  - The names of the object affected by the transaction (the name of the table).
  - The "before" and "after" values of the fields being updated.
  - Pointers to the previous and next transaction log entries for the same transaction.
- The ending (COMMIT) of the transaction.

4. Concurrency control refers to managing the simultaneous execution of transactions in a multiuser database system. It ensures the serializability of transactions in a multiuser database environment. Concurrency control is an extremely important function of an RDBMS because transactions executing at the same time can cause data integrity and consistency problems.
5. The three main problems are lost updates, uncommitted data and inconsistent retrievals.

The **lost update** problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost.

Uncommitted data occurs when two transactions, T1 and T2, are executed at the same time and the first transaction is rolled back after the second transaction has already accessed the uncommitted data.

Inconsistent retrievals occur when a transaction accesses data before and after another transaction(s) finish working with that data.

6. The **scheduler** is a special DBMS process that establishes the order in which the operations within which concurrent transactions are executed.
7. **Lock granularity** specifies the level of lock use. Locking can take place in the following levels: database, table, page, row or field (attribute).

Database Level

In a **database-level lock**, the whole database is locked down by a single transaction. This prevents another transaction from using any tables in the database while the current transaction is being executed. This is good for batch processes, but not for multiuser DBMSs - imagine the chaos which would follow if every Facebook user had to wait in a queue while one person updates his profile! Clearly this is because it would be very slow if thousands of transactions had to wait for the previous transaction to be complete before the next one could reserve the database.

Table Level

In a **table-level lock**, the whole table is locked. This prevents access to any row in the table by a transaction while another transaction is using the table. If a transaction needs access to many tables, each table may be locked. However, two transactions can access the same database as long as they access different tables.

Table-level locks are less restrictive that database-level locks. However they cause traffic jams when many transactions are waiting to access the same table. They are not suitable for multiuser DBMSs.

Page Level

In a **page-level lock**, the DBMS locks an entire diskpage. A **diskpage**, or **page**, is the same as a *diskblock*. A **diskblock** is a section of a disk that can be accessed directly. A page has a fixed size, such as 4K, 8K or 16K. A table can extend across many pages. A page can contain many rows of one or more tables. Page-level locks are the most frequently used multiuser DBMS locking method.

Row Level

A **row-level lock** is less restrictive that the other locks. The DBMS allows concurrent transactions to access different rows of the same table even when the rows are located on the same page. It improves the availability of data. However, it requires high overhead to manage. This is because a lock exists for each row in a table of the database involved in a conflicting transaction. Modern DBMSs automatically escalate a lock from a row-level to a page-level lock when the application requests many locks on the same page.

Field Level

The **field-level lock** is the least restrictive lock type – it allows parallel transactions to access the same row as long as they use different fields (attributes) within that row. It results in the most flexible multiuser access. However, it is rarely implemented in a DBMS. This is because it requires a very high level of computer overhead and also because the row-level lock is more useful.

Unit 8:

Developing an object orientated computer program in Java to manipulate a relational database

Unit Learning Outcomes

CONTENT LIST	LEARNING OUTCOMES OF THIS UNIT:
8.1 Introduction	• Be able to explain what a CRUD application is
8.2 Getting the required software	• Know what Netbeans is, and how to download and install it
8.3 Creating the database	• Be able to design and implement a simple relational database using Netbeans • Create a visual, object oriented computer program in Java to manipulate a relational database

Prescribed and Recommended Textbooks/Readings

- Rob, Peter et al, Database Systems – Design, Implementation & Management International Edition ISBN: 978-1-84480-732-1

8.1 Introduction

In this chapter you will put all the skills you've gained in this and the previous modules into practice by creating your own database from the ground up. Added to that, you will create a Java CRUD application to interact with your database. A CRUD application is an application that interacts with a database via four operations:

1. C – Create information in the database
2. R – Read data from the database
3. U – Update data in the database
4. D – Delete information from a database

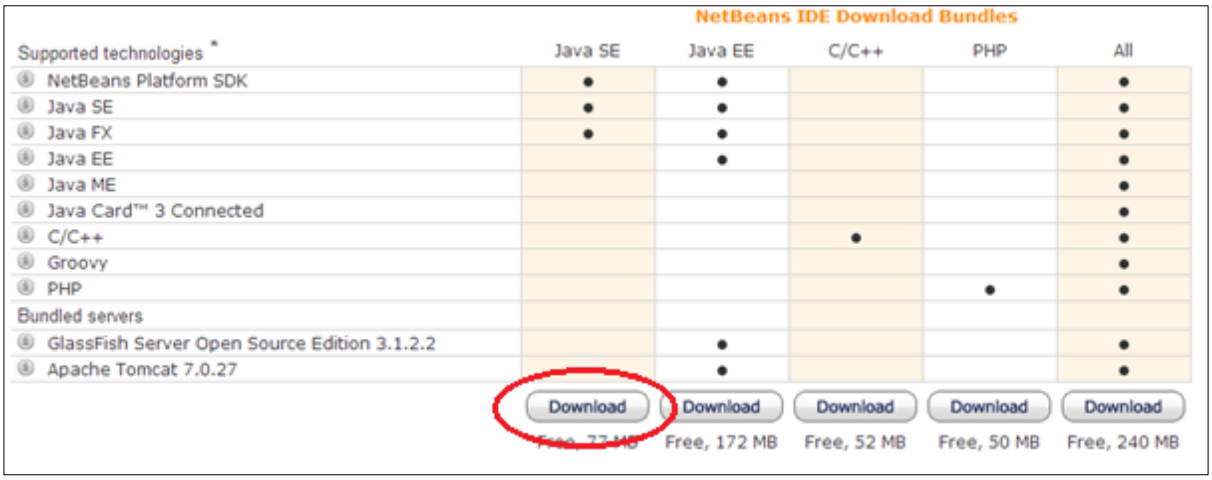
As you can see, the word CRUD is actually an acronym that is derived from the words Create, Read, Update and Delete.

8.2 Getting the required software

In order to create your database as well as your Java application, you will need a software package called NetBeans, which is a well-known Java IDE (Integrated Development Environment). Netbeans is a free, open source application that is downloadable from the following web page:

http://netbeans.org/downloads/

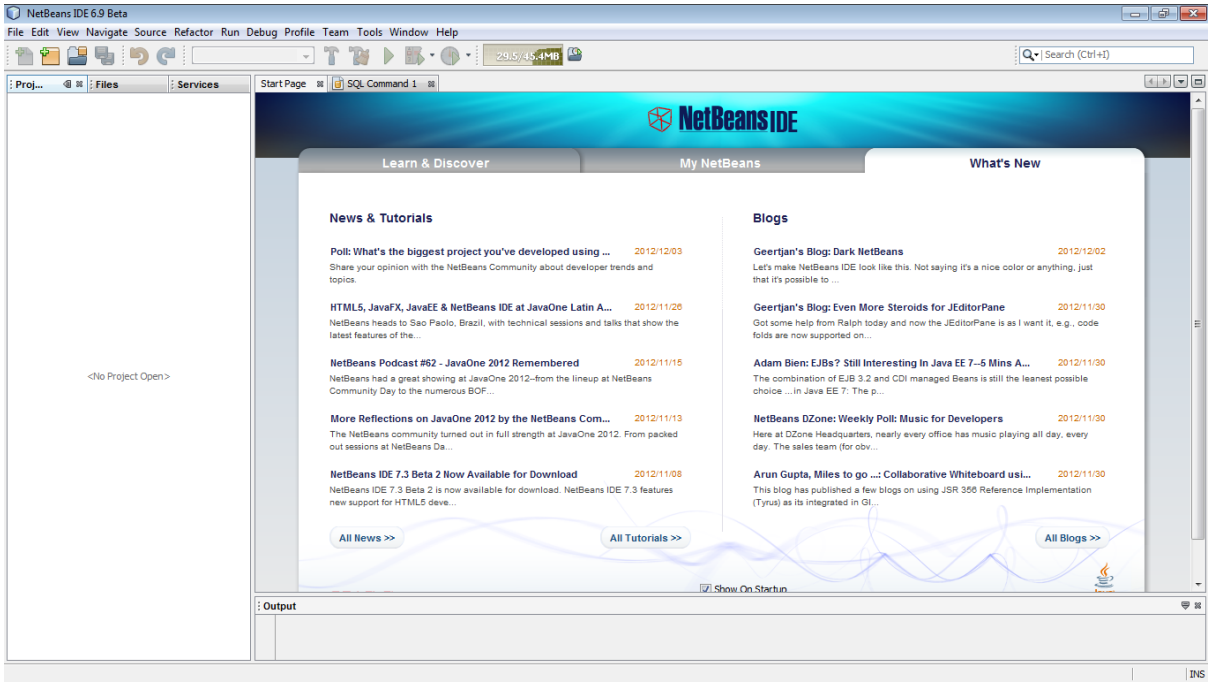
When you type the above link into your browser address bar, you will be directed to the Netbeans download page where you will see the following table:



Click on the first **Download** button, as shown in the above diagram. This will begin the Netbeans download. Note that it is a very large download totaling 77MB, so it will take some time.

Once you have downloaded Netbeans, install it on your machine just as you would install any other software - simply follow the instructions.

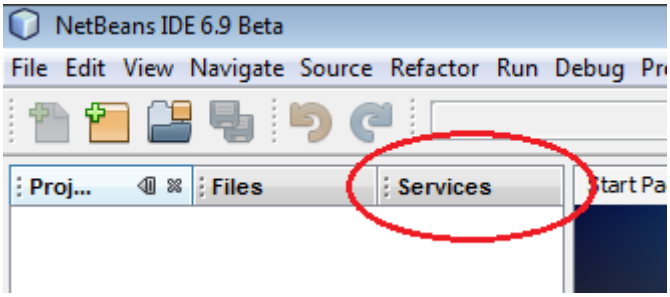
Once you've installed Netbeans, open it. You will see the following screen:



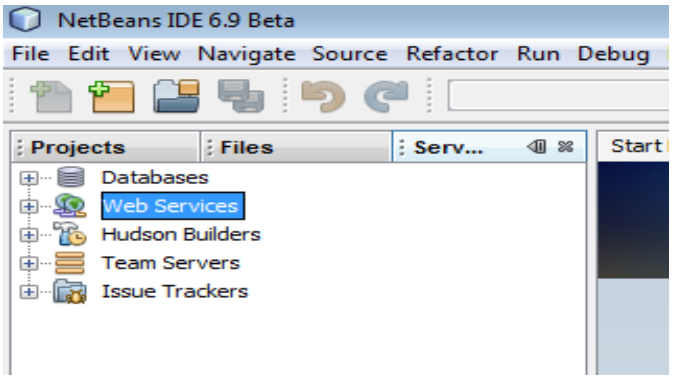
This is the Netbeans home screen.

8.3 Creating the database

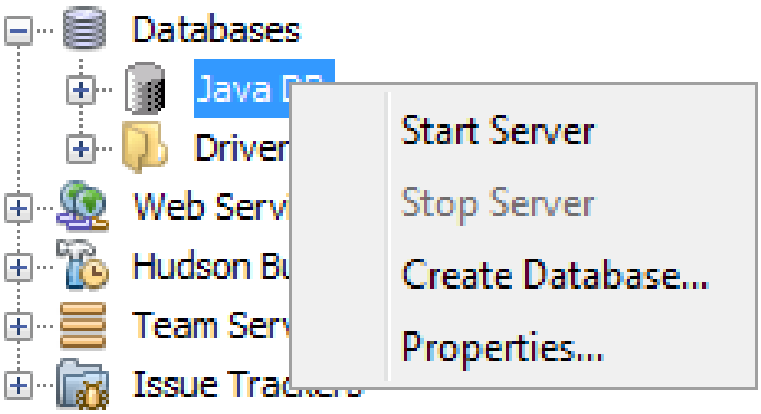
To begin creating your database, click on the Services button (near the top-left of the screen) to open the **Services** window, as shown below:



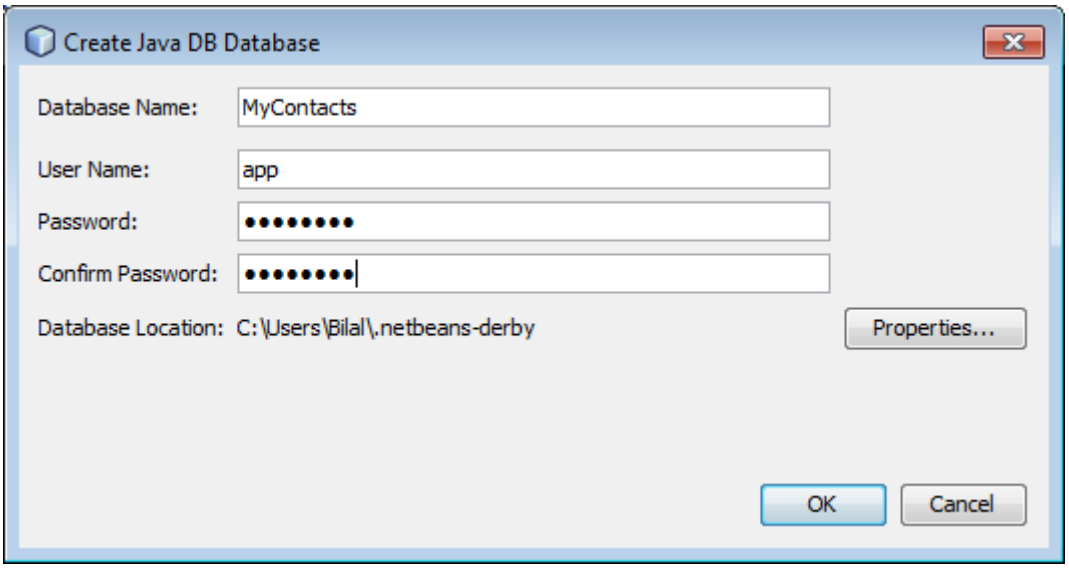
The **Services** window looks like this:



Click on the [+] next to **Databases**, and then right-click on **JavaDB**. A pop-up menu will appear as follows:



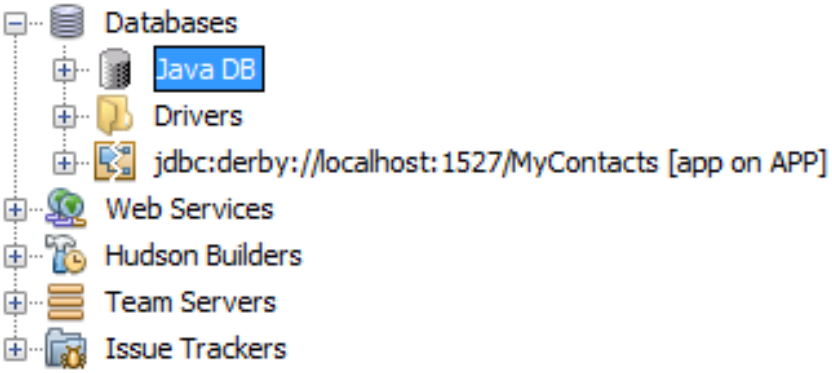
Click on **Create Database**. The following window will appear:



Our database will have a single table to store contact details, similar to the contacts app on a cellular phone. Each row of the contacts table will store the name, surname, telephone number, cellphone number and email address of our contacts.

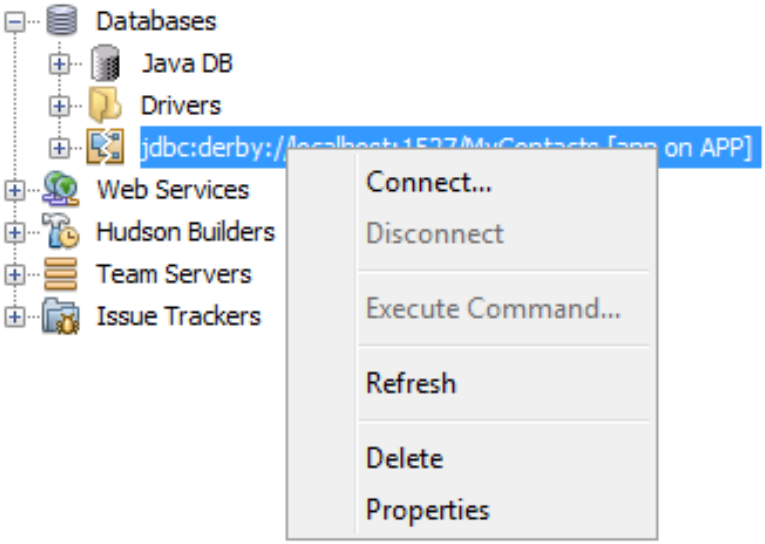
Fill in the fields exactly as shown above. You may set the password to anything you wish, but you will need to remember it.

Once you've entered all the fields, click **OK**. A new, blank database will be created. The **Services** window should now look as follows:

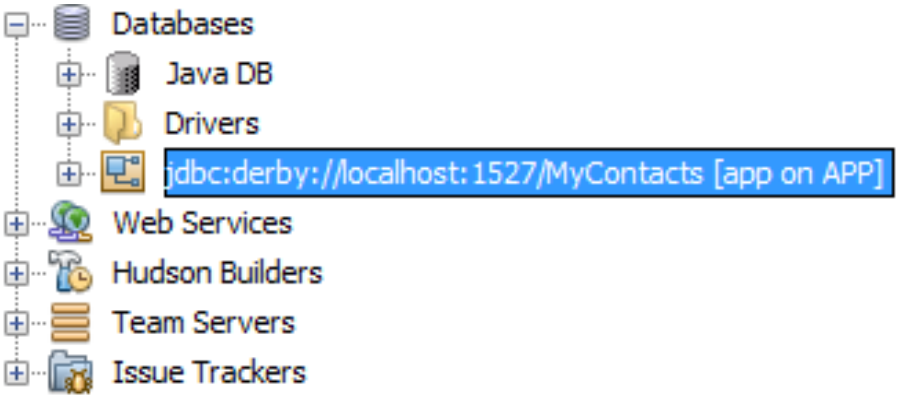


Right-click on **jdbc:derby://localhost:1527/MyContacts [app onAPP]**

The following pop-up menu will appear:



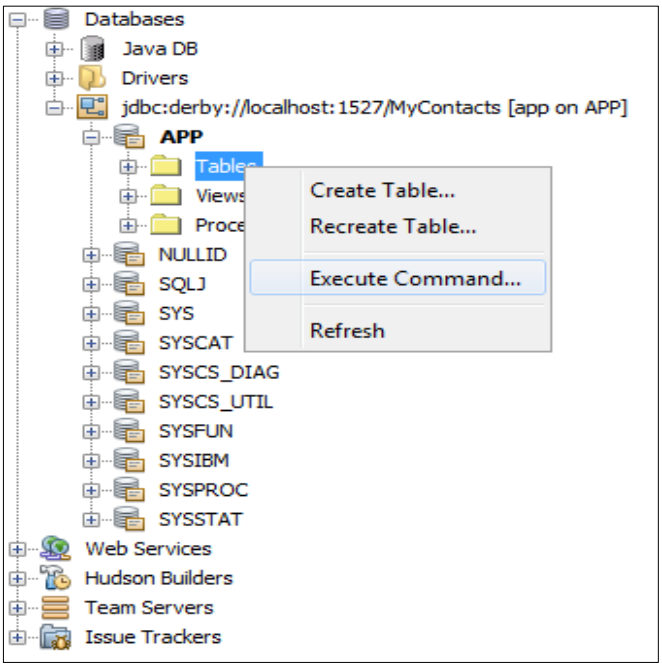
Click on connect. The **Services** window should now look as follows:



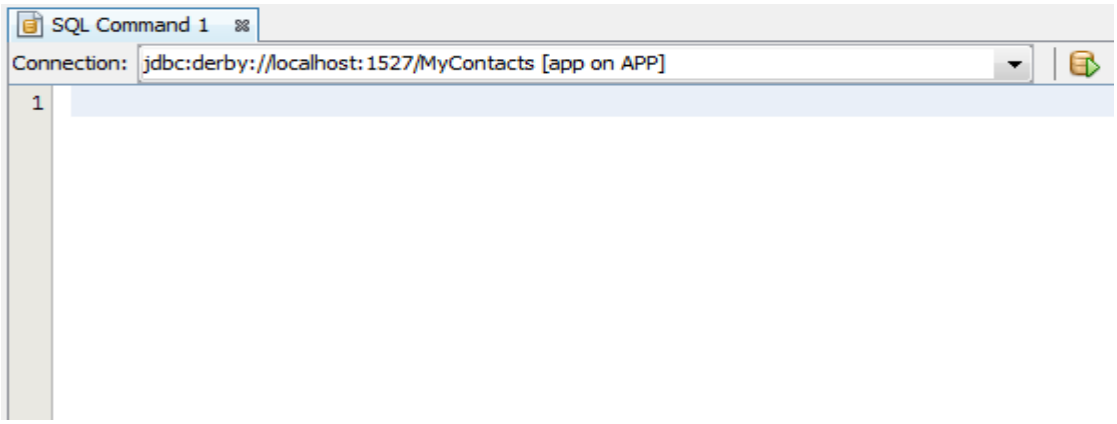
Notice that the icon next to **jdbc:derby://localhost:1527/MyContacts [app onAPP]** has changed from  to  . This indicates that Netbeans is now connected to your database. You may now create a table in your database.

Creating a table in the database

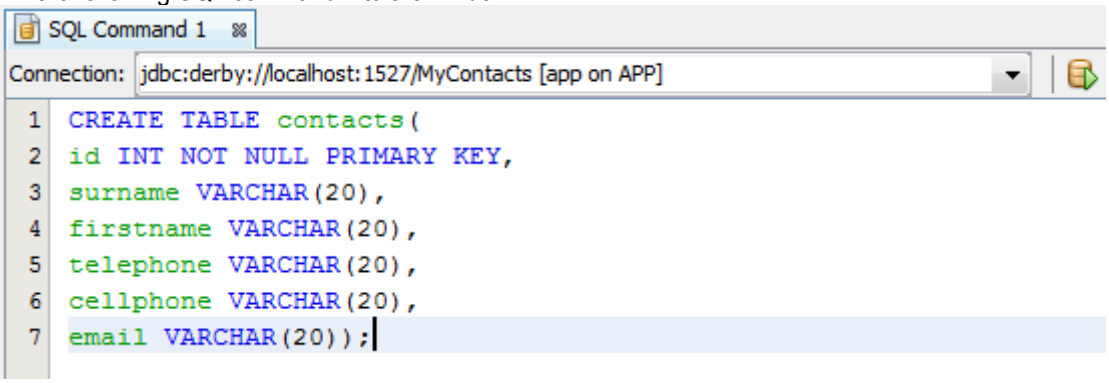
Click on the [+] next to **jdbc:derby://localhost:1527/MyContacts [app onAPP]** and then the [+] next to **APP** in order to expand the submenus below them, as shown below. Then, right-click on **Tables**. From the pop-up menu, click on **Execute Command…**



The SQL Command 1 tab will appear in the main section of the screen:

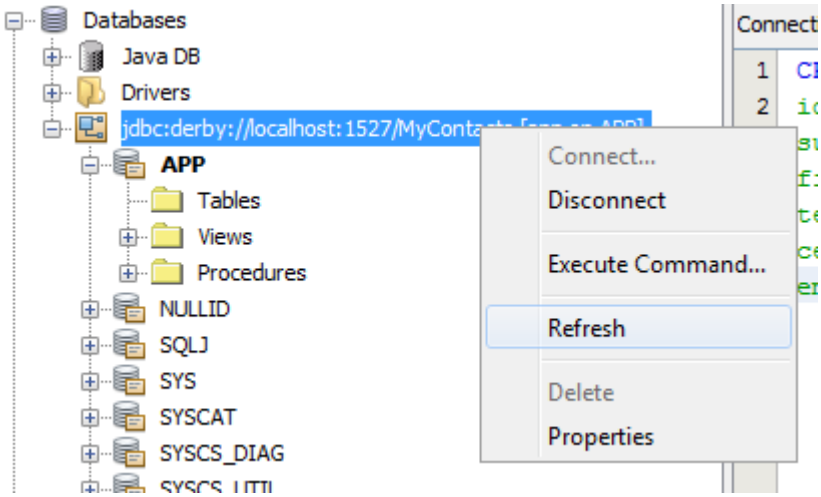


Type in the following SQL command into the window:

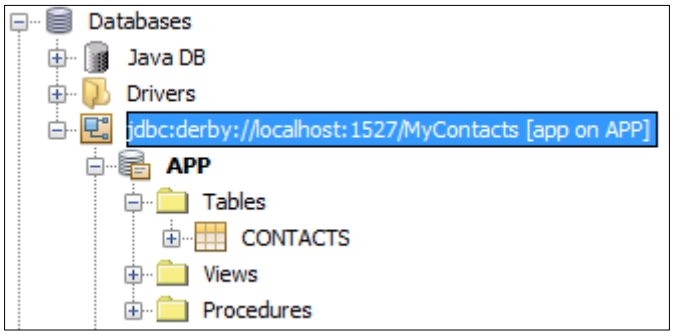


Check that you've entered the SQL command exactly as shown above, and then press the  button.

If you've typed it in correctly, your new table will be created. To see your table, right-click on **jdbc:derby://localhost:1527/MyContacts [app onAPP]** and then click on the **Refresh** option.



You will then be able to see your newly-created table called **CONTACTS** under **APP** → **Tables** as shown below:

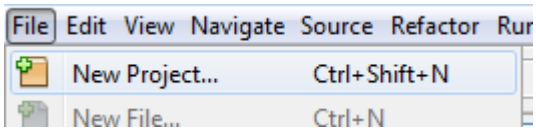


If you can see the **CONTACTS** table as shown above, then you can move on because you've successfully created the table.

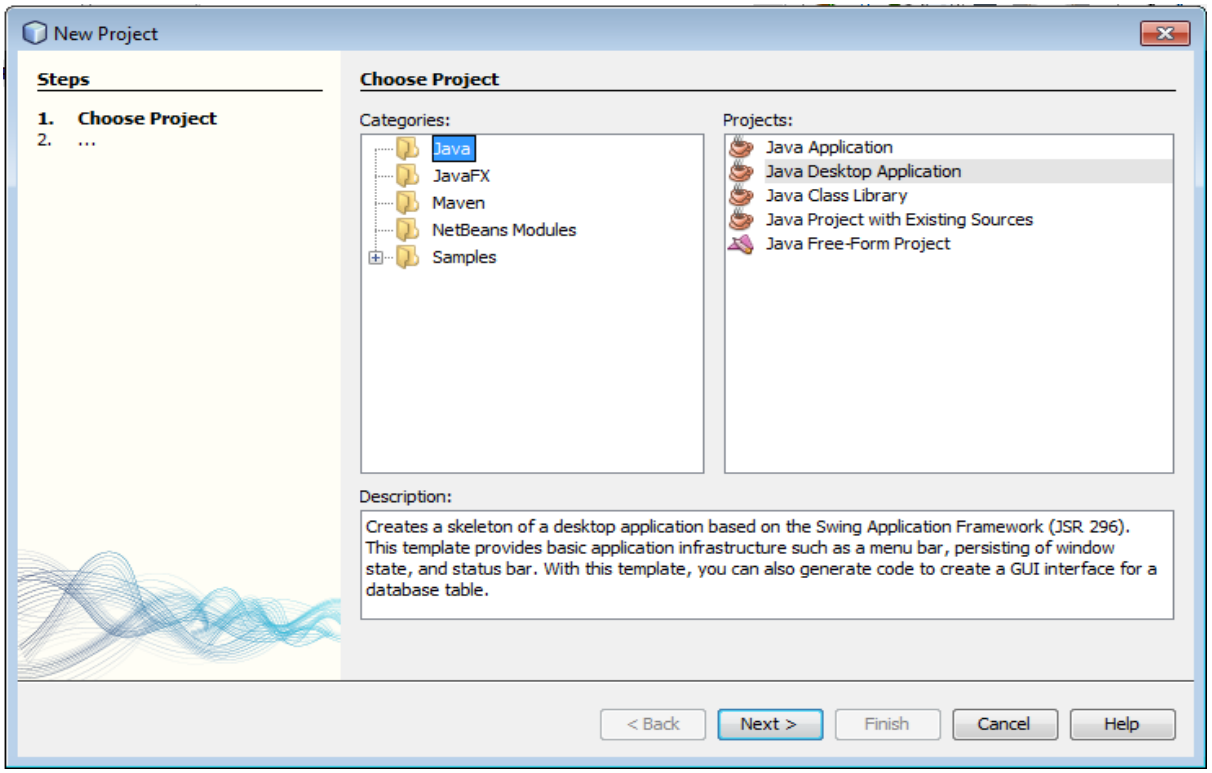
Now that our database has been created, we will go on to the next step, which is to create the Java application to interact with the database.

Creating a Java application to manipulate the database

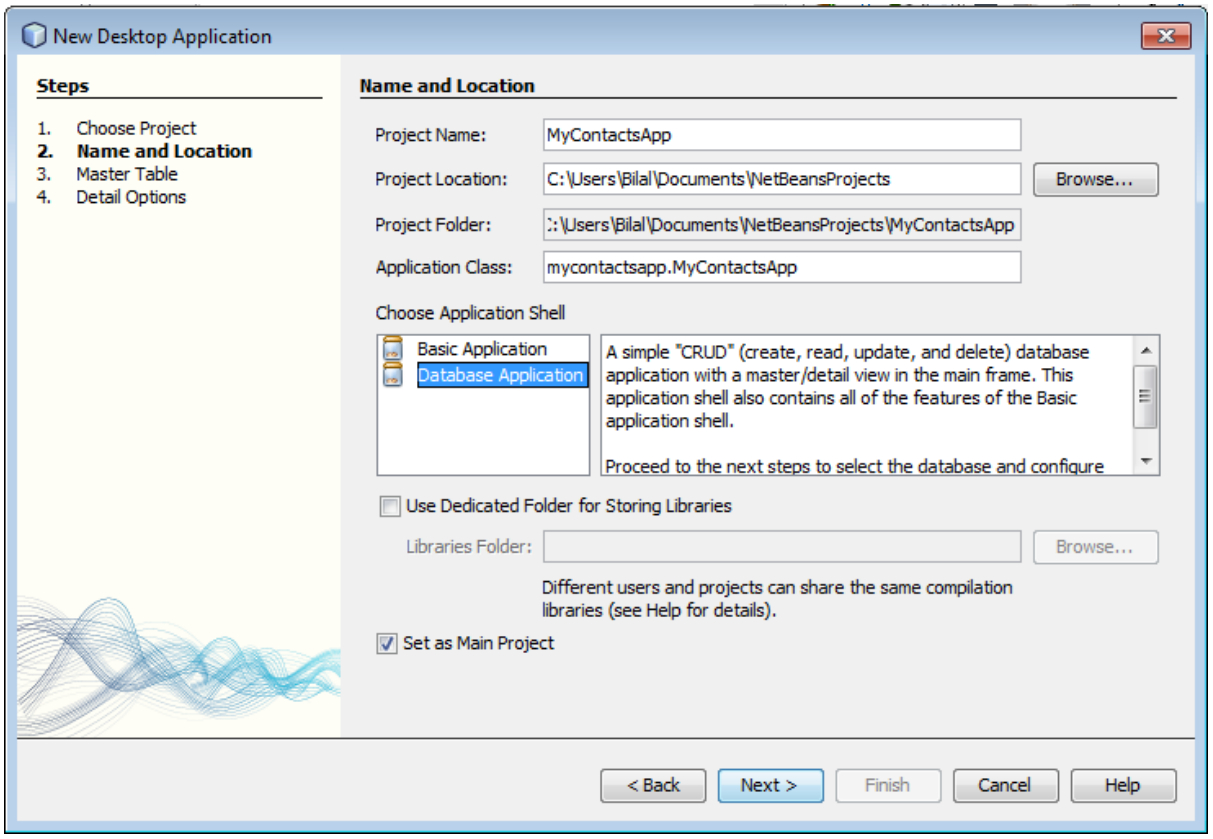
Click on **File** → **New Project…** as shown below:



The **New Project** window will appear:

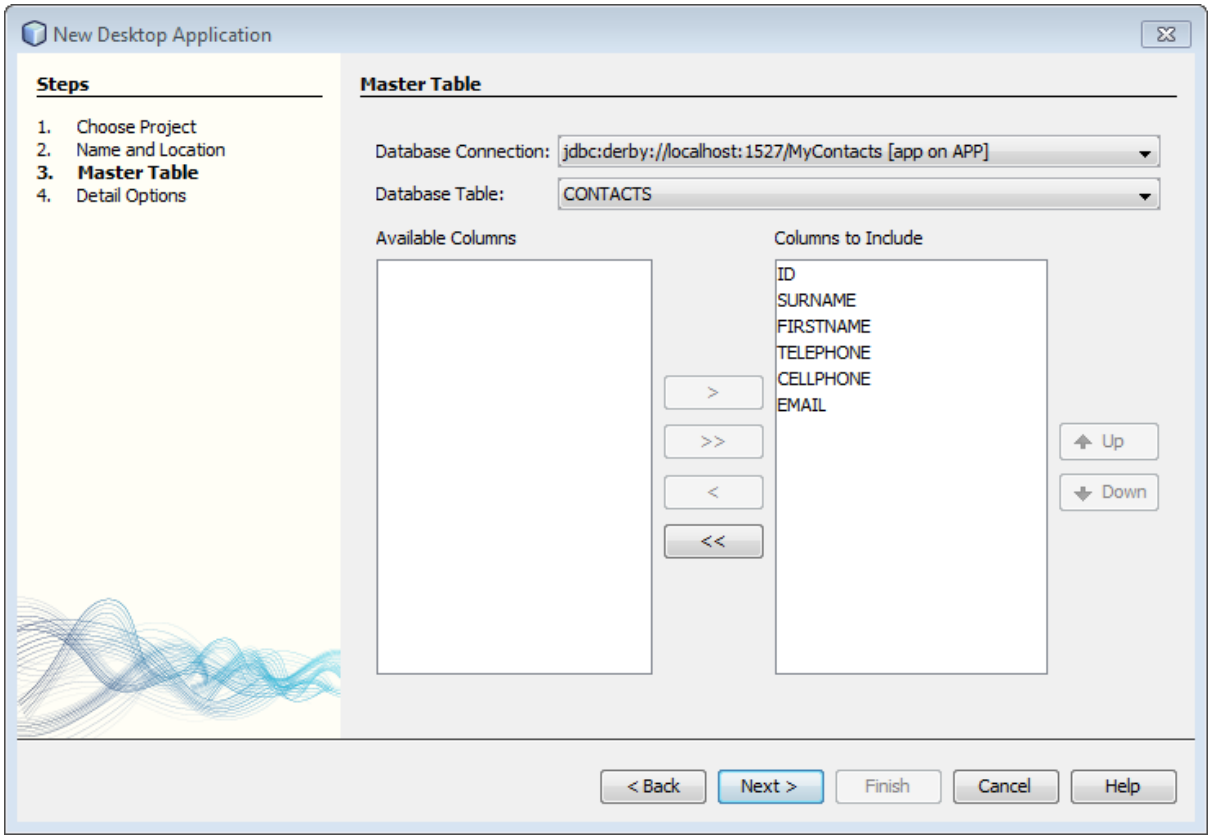


Under **Categories**, choose **Java**; under **Projects**, choose **Java Desktop Application**. Then, click **Next**.

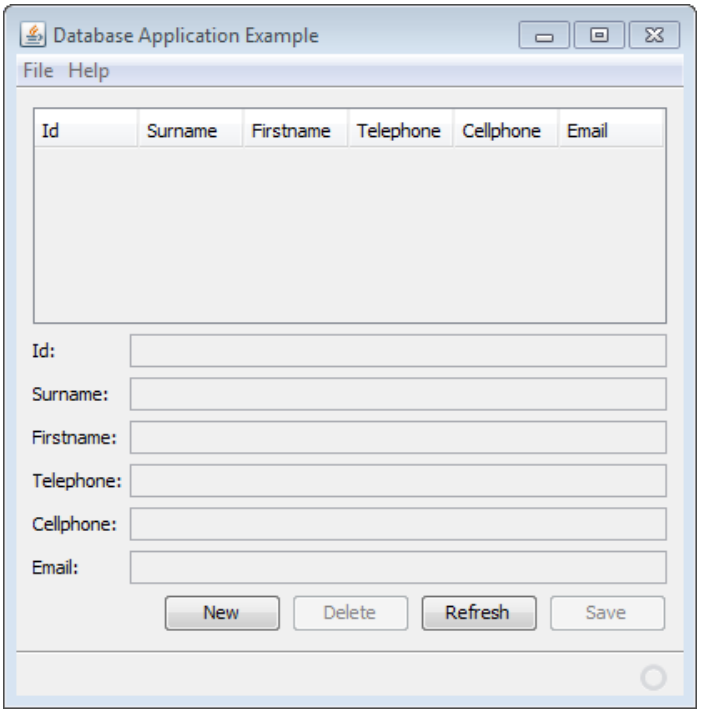


In the **New Desktop Application** window set your Project Name to **MyContactsApp** as shown in the previous screenshot. In the **Choose Application Shell** box, make sure that the **Database Application** option is selected. Click **Next**.

In the next window, click on the dropdown button next to Database Connection and select the **jdbc:derby://localhost:1527/MyContacts [app onAPP]** option. All the columns from the contacts table will appear under **Columns to Include**, as shown below:

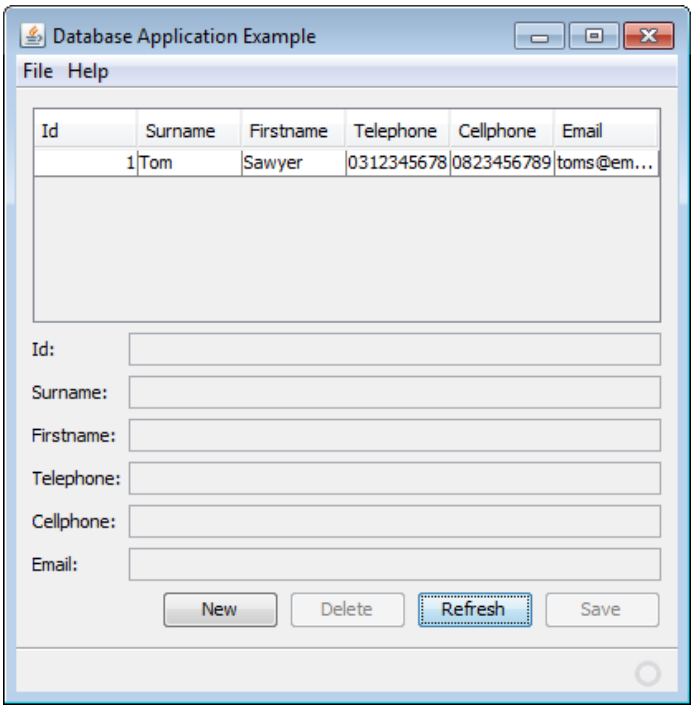


Click **Next** and then **Finish**. Your application will be created. To run the app, click on  in the toolbar.



Your application will then run, and the main screen will be displayed, as shown in the next page:

To insert a new contact into the database, click on **New** and then type in the information into the text boxes. Then click **Save**. Your new contact will then appear at the top of the screen. If it does not appear, click the **Refresh** button.



You have developed an object-orientated computer program in Java to manipulate a relational database.

Conclusion

In this module you've learned about databases, which form the storage mechanism for the bulk of the world's data. You have become familiar with what databases are and how they work, and you have also been equipped with the knowledge and skills to design and implement fairly complex databases. These skills will undoubtedly prove extremely useful to you as a future Information Technology professional.

Databases which live and run on the internet have revolutionized the internet. From Google's massive search databases to Facebook's databases which hold user information, to databases owned by banks and other financial institutions which dispense valuable information, these marvels of technology are at work 24 hours a day, and have a tremendous impact on our lives.

In the next module, Informatics 2B, you will be introduced to yet another way that web databases have had a tremendous impact on global business and on our lives: e-Business. As with so many other things, the internet has revolutionized the way we do business and the way we buy and sell things; and you will be shown exactly how this has happened. You will also become familiar with the mechanisms behind e-business so that you may be able to contribute to your organization by taking their operations online. In fact, with the knowledge you will gain, you may even start your own e-commerce business.

MANCOSA
Member of
HONORIS UNITED UNIVERSITIES

MODULE REVIEW AND FEEDBACK FORM

Name of Student:

Programme:

Module:

MANCOSA is committed to continuous upgrading and improving the content and presentation of our module guides. Your constructive feedback is appreciated.

Please e-mail to : Curriculum Development and Review

E-mail : modulefeedback@mancosa.co.za