Introduction to Databases
Copyright

This course has been developed as part of the collaborative advanced ICT course development project of the Commonwealth of Learning (COL). COL is an intergovernmental organization created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources and technologies.

The Open University of Tanzania (OUT) is a fully fledged, autonomous and accredited public University. It offers its certificate, diploma, degree and postgraduate courses through the open and distance learning system which includes various means of communication such as face-to-face, broadcasting, telecasting, correspondence, seminars, e-learning as well as a blended mode. The OUT’s academic programmes are quality-assured and as centrally regulated by the Tanzania Commission for Universities (TCU).

© 2016 by the Commonwealth of Learning and The Open University of Tanzania. Except where otherwise noted, Introduction to Databases is made available under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License: https://creativecommons.org/licenses/by-sa/4.0/legalcode.

For the avoidance of doubt, by applying this licence the Commonwealth of Learning does not waive any privileges or immunities from claims that it may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes or laws of any jurisdiction. The ideas and opinions expressed in this publication are those of the author/s; they are not necessarily those of Commonwealth of Learning and do not commit the organisation.
Acknowledgements

The Open University of Tanzania (OUT), Faculty of Science, Technology and Environmental Studies, (FSTES) and the ICT Department wish to thank those below for their contribution to the production of this course material and video lectures:

Authors:
Ms. Grace W. Mbwete (Assistant Lecturer, ICT Department, OUT)
Mr. Elia A. Lukwaro (Tutorial Assistant, ICT Department, OUT)

Copy Editor:
Mr. Justin Kimaro (Principal Editor, IEMT, OUT)

Reviewers:
Ms. Lilian C. Mutalemwa (Assistant Lecturer, ICT Department, OUT)
Ms. Regina Monyemangene (Instructional Designer, IEMT, OUT)

Video Production:
Mr. Othman Mwinchoum (Multimedia studio, IEMT, OUT)
Ms. Grace W. Mbwete (Assistant Lecturer, ICT Department, OUT)
Ms. Lilian C. Mutalemwa (Assistant Lecturer, ICT Department, OUT)

This course has been developed with the support of the Commonwealth of Learning, Canada.
# Contents

## About this Course Material

- How this Course Material is structured ................................................................. 1

## Course overview

- Welcome to: Introduction to Databases ...................................................................... 3
- Course Overview Video ............................................................................................... 3
- Is this course for you? .................................................................................................. 3
- Course outcomes ......................................................................................................... 4
- Timeframe .................................................................................................................... 4
- Need help? ................................................................................................................... 5
- Assessment .................................................................................................................. 5

## Getting around this Course Material

- Margin icons ............................................................................................................... 6

## Unit 1

### Introduction to Database Systems

- Introduction ................................................................................................................ 7
- Unit Outcomes ............................................................................................................. 7
- Terminologies .............................................................................................................. 7

1.1 Databases: Definition and Characteristics .............................................................. 8

1.2 Functions of a Database ........................................................................................... 10

1.3 Database Usage and Environment .......................................................................... 10

1.4 Database Users ....................................................................................................... 11

1.5 Classification of DBMSs ....................................................................................... 12

### Video lecture:

- Activity ....................................................................................................................... 15

### Unit summary

- Review Questions ....................................................................................................... 16

### References and Further Reading

- Attribution .................................................................................................................. 17

## Unit 2

### Database History

- Introduction ................................................................................................................ 19
- Unit Outcomes ............................................................................................................. 19
- Terminologies .............................................................................................................. 19

2.1 Before the Advent of Databases .............................................................................. 20

2.2 Database Approach ............................................................................................... 22
## Contents

- Video lecture: ................................................................. 105
- Activity ........................................................................ 106
- Unit summary ................................................................. 106
- Review Questions .......................................................... 107
- References and Further Reading .................................... 107
- Attribution ...................................................................... 108

### Unit 9  
109

- Functional Dependencies .............................................. 109
  - Introduction ................................................................ 109
  - Unit Outcomes .......................................................... 109
  - Terminologies ............................................................ 109
  - 9.1 Functional Dependency ......................................... 110
  - 9.2 Rules of Functional Dependencies ....................... 110
  - 9.2 Dependency Diagram ........................................... 115
- Video lecture: ................................................................. 116
- Activity ........................................................................ 116
- Unit summary ................................................................. 117
- Review Questions .......................................................... 117
- References and Further Reading .................................... 119
- Attribution ...................................................................... 119

### Unit 10  
120

- Introduction to Data Normalization ..................................... 120
  - Introduction ................................................................ 120
  - Unit Outcomes .......................................................... 120
  - Terminologies ............................................................ 120
  - 10.1 Overview .............................................................. 121
  - 10.2 Normal Forms ..................................................... 121
  - 10.3 Normalization and Database Design .................. 128
- Video lecture: ................................................................. 128
- Activity ........................................................................ 129
- Unit summary ................................................................. 130
- Case Study ....................................................................... 130
- References and Further Reading .................................... 133
- Attribution ...................................................................... 134

### Unit 11  
135

- Introduction to SQL .......................................................... 135
  - Introduction ................................................................ 135
  - Unit Outcomes .......................................................... 135
  - Terminologies ............................................................ 135
  - 11.1 Introduction to SQL ............................................. 136
  - 11.2.2 Data Definition Language (DDL) Commands ..... 136
  - 11.3 Table Constraints ............................................... 143
  - 11.4 User Defined Types ............................................. 147
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Outcomes</td>
<td>191</td>
</tr>
<tr>
<td>Terminologies</td>
<td>191</td>
</tr>
<tr>
<td>14.1 Overview</td>
<td>192</td>
</tr>
<tr>
<td>14.2 Software Development Life Cycle (SDLC)</td>
<td>192</td>
</tr>
<tr>
<td>14.3 Database Life Cycle</td>
<td>194</td>
</tr>
<tr>
<td>Video lecture</td>
<td>204</td>
</tr>
<tr>
<td>Activity</td>
<td>205</td>
</tr>
<tr>
<td>Unit summary</td>
<td>206</td>
</tr>
<tr>
<td>Review Questions</td>
<td>206</td>
</tr>
<tr>
<td>References and Further Reading:</td>
<td>207</td>
</tr>
<tr>
<td>Attribution</td>
<td>207</td>
</tr>
</tbody>
</table>

**Video 20 – Review Questions and Activities**

208
About this Course Material

Introduction to Databases has been produced by The Open University of Tanzania in collaboration with Commonwealth of Learning. This Course Material is structured as outlined below:

How this Course Material is structured

The course overview

The course overview gives you a general introduction to the course. Information contained in the course overview will help you determine

- If the course is suitable for you.
- What you will already need to know.
- What you can expect from the course.
- How much time you will need to invest to complete the course.

The overview also provides guidance on

- Study skills.
- Where to get help.
- Course assignments and assessments.
- Activity icons.
- Units.

We strongly recommend that you read the overview carefully before starting your study.

The course content

The course is broken down into units. Each unit comprising:

- An introduction to the unit content.
- Unit outcomes.
- New terminology.
- Core content of the unit with a variety of learning activities.
- A unit summary.
- Review questions, case studies and assignments, as applicable.
- Reference and Further Reading.
About this Course Material

- Attribution.
- Links to video lectures for each unit are provided at the end of this course material.

Resources

For those interested in learning more on this subject, we provide you with a list of additional resources at the end of each unit, these may be books, articles or websites. Links to video lectures for each unit are provided at the end of this course material.

Your comments

After completing Introduction to Databases we would appreciate it if you could take a few moments to give us your feedback on any aspect of this course. Your feedback might include comments on:

- Course content and structure
- Course reading materials and resources
- Course activities
- Video lectures which are provided at the end of this course material
- Course review questions
- Course duration
- Course support (assigned tutors, technical help, etc.)

Your constructive feedback will help us to improve and enhance this course in future.
Course overview

Welcome to: Introduction to Databases

This course will equip students with ICT skills which include theory and practical of fundamental concepts of database management systems, data modelling techniques, data normalization, Structured Query Language (SQL) and finally, database development process. At the end of this course, you should be able to take a real world case scenario and create a database that adhere to a user requirements and required functions.

Course Overview Video

https://tinyurl.com/h4ktsab

Is this course for you?

This course is intended for people who are already computer literate. The course aims are:

- To introduce learners to the requisite theory and practice of database technology and the applications of the technology in generic and specific domains.
- To enable the learners to learn and apply methodologies for conceptual, logical and physical database design.
Course overview

- To enable the learner to acquire skills in solving business problems using the fundamentals of database modeling, enterprise analysis and design.
- To introduce learners to implementation and management issues as well as database definition and manipulation languages.

Course outcomes

Upon completion of Introduction to Databases, learners should be able to:

- Explain the basic definition and characteristics of databases.
- Compare and contrast different roles of database end users.
- Differentiate between various data models.
- Design physical database schema for given user requirements.
- Develop an entity-relationship model based on user requirements.
- Normalize a set of relational schema to a third normal form.
- Describe and use data definition language commands (SQL).
- Describe and use data manipulation commands (SQL).
- Describe how to join tables for use in SQL.
- Design, Implement and Test a database according to given user requirements.

Timeframe

This is a one-year course with an exit mode in between.

This course requires timeframe that depends on individual institution’s mode of delivery.

A minimum standard of delivery should be 20 weeks of formal lectures, 12 weeks of supervised laboratory tutorials and 4 weeks of unsupervised directed learning.

Self-study time is 10 hours per-week.
Need help?

This course is offered at The Open University of Tanzania, ICT department.
If you need help regarding this course, please contact:

Head of Department, ICT department
The Open University of Tanzania
P. O. Box 23409,
Kinondoni,
Dar Es Salaam,
Tanzania
Phone: +255 22 2668992
Fax: +255 22 2668756
Email: hod.ict@out.ac.tz
Website: www.out.ac.tz

Assessment

There are activities, case studies, assignments and review questions in the units of this course. All these learner’s activities are assessed in three modalities

- Peer – review
- Self – assessment
- Instructor – marked assessment

NB: Review questions are for self – assessment
Getting around this Course Material

Margin icons

While working through this Course Material you will notice the frequent use of margin icons. These icons serve to “signpost” a particular piece of text, a new task or change in activity; they have been included to help you to find your way around this Course Material.

A complete icon set is shown below. We suggest that you familiarize yourself with the icons and their meaning before starting your study.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Assessment</th>
<th>Assignment</th>
<th>Case study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion</td>
<td>Group activity</td>
<td>Help</td>
<td>Note it!</td>
</tr>
<tr>
<td>Outcomes</td>
<td>Reading</td>
<td>Reflection</td>
<td>Study skills</td>
</tr>
<tr>
<td>Summary</td>
<td>Terminology</td>
<td>Time</td>
<td>Tip</td>
</tr>
</tbody>
</table>
Unit 1

Introduction to Database Systems

Introduction

In this unit, we are going to be introduced to the basic concepts of database systems. This is an introductory unit where we are going to learn about the definition of a database, its characteristics, classification, functions and the user types which are involved in using the database.

Unit Outcomes

Upon completion of this unit you should be able to:

- Describe what a database is and how it functions.
- Outline various properties of a database.
- Explain the roles of different database end users.
- Compare and contrast the different databases based on their classification.

Terminologies

**Database:** Organised set of data in a computer.

**DBMS:** Database Management System.

**Data:** Raw facts.

**Information:** Processed data.

**Centralized DB:** Centralized Database is a database which is located and managed in one site.

**DBA:** Database Administrator - performs all activities related to maintaining a successful database environment.
1.1 Databases: Definition and Characteristics

1.1.1 What is Database?

A database is a shared collection of related data used to support the activities of a particular organization. A database can be viewed as a repository of data that is defined once and then accessed by various users as shown in Figure 1.1.

![Database System Diagram](image)

Figure 1.1: A database is a repository of data, by A. Watt.

1.1.2 Characteristics of a Database

There are a number of characteristics that distinguish the database approach with the file-based approach. The following are the characteristics (features) and benefits of the database systems:

i) **Self-Describing Nature of a Database System**

A database system is referred to as *self-describing* because it not only contains the database itself, but also *metadata* which defines and describes the data and relationships between tables in the database. This information is used by the DBMS software or database users if needed. This separation of data and information about the data makes a database system totally different from the traditional file-based system in which the data definition is part of the application programs.
ii) **Insulation between Program and Data**

In the file based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well. On the other hand, in the database approach, the data structure is stored in the system catalog and not in the programs. Therefore, one change is all that’s needed.

iii) **Support multiple views of data**

A view is a subset of the database which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or a group of users.

iv) **Multiuser system**

A multiuser database system must allow multiple users access to the database at the same time. As a result, the multiuser DBMS must have concurrency control strategies to ensure several users access to the same data item at the same time, and to do so in a manner that the data will always be correct – data integrity.

v) **Control Data Redundancy**

In the Database approach, ideally each data item is stored in only one place in the database. In some cases redundancy still exists so as to improve system performance, but such redundancy is controlled and kept to minimum.

vi) **Data Sharing**

The integration of the whole data in an organization leads to the ability to produce more information from a given amount of data.

vii) **Enforcing Integrity Constraints**

DBMSs should provide capabilities to define and enforce certain constraints such as data type and data uniqueness.

viii) **Restricting Unauthorized Access**

Not all users of the system have the same accessing privileges. DBMSs should provide a security subsystem to create and control the user accounts.
ix) **Data Independence**

System data (Meta Data) descriptions are separated from the application programs. Changes to the data structure is handled by the DBMS and not embedded in the program.

x) **Transaction Processing**

The DBMS must include concurrency control subsystems to ensure that several users trying to update the same data do so in a controlled manner. The results of any updates to the database must maintain consistency and validity.

xi) **Providing backup and recovery facilities**

If the computer system fails in the middle of a complex update process, the recovery subsystem is responsible for making sure that the database is restored to the stage it was in before the process started executing.

### 1.2 Functions of a Database

Today, databases are widely being used by different organizations whereby application depends on their specific requirements and business needs. In summary, this section will outline functions of a database so that there is greater understanding of why an individual or an organization would want to use a database management system. These functions includes but not limited to:

- To organize and store data.
- To provide facilitation for analysis and modeling.
- To process given data and turn it into information
- To explore data using exploratory techniques
- To support organizational activities

### 1.3 Database Usage and Environment

Database usage is almost everywhere in different aspects of everyday life. The following are some of the applications in which databases are used in carrying out organizational activities:

- Banking – assist in carrying out banks’ customer’s financial transactions.
• Airlines – provide flexible means of flight scheduling and reservations.
• Universities – platform for registration and grades recording for students.
• Sales – storing customer, products and purchases details.
• Manufacturing – organizing data on production, inventory, orders and the whole supply chain.

1.4 Database Users

These are people whose jobs require access to a database for querying, updating and generating reports. An end user might be one of the following:

1.4.1 Application User
The application user uses the existing application programs to perform their daily tasks.

1.4.2 Sophisticated user
Sophisticated users are those who have their own way of accessing the database. This means they do not use the application program provided in the system. Instead, they might define their own application or describe their need directly by using query languages. These specialized users maintain their personal databases by using ready-made program packages that provide easy-to-use menu driven commands, such as MS Access.

1.4.3 Application Programmers
Application Programmers are technical personnel who implement specific application programs to access the stored data. This kind of user needs to be familiar with the DBMSs to accomplish their task.

1.4.4 Database Administrators
Database Administrator(s) can be a one person or a group of people in the organization who is responsible for authorizing the access to the database, monitoring its use and managing all the resource to support the use of the whole database system.
1.5 Classification of DBMSs

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution as shown follows:

1.5.1 Data Model
The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and MySQL support this model. Other traditional models, such as hierarchical data models and network data models are still used in industry mainly on mainframe platforms. However, they are not commonly used due to their complexity. These are all referred to as traditional models because they preceded the relational model.

In recent years, the newer object-oriented data models were introduced. This model is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-oriented databases are different from relational databases which are table-oriented. Object-oriented database management systems (OODBMS) combine database capabilities with object-oriented programming language capabilities.

The object-oriented models have not caught on as expected so are not in widespread use. Some examples of object-oriented DBMSs are O2, ObjectStore and Jasmine.

1.5.2 Number of Users
A DBMS can be classification based on the number of users it supports. It can be a single-user database system, which supports one user at a time, or a multiuser database system, which supports multiple users concurrently.

1.5.3 Distribution
There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.
i) Centralized systems

With a *centralized database system*, the DBMS and database are stored at a single site that is used by several other systems too. This is illustrated in Figure 1.2.

![Figure 1.2: A centralized database system, by A. Watt](image)

In the early 1980s, many Canadian libraries used the GEAC 8000 to convert their manual card catalogues to machine-readable centralized catalogue systems. Each book catalogue had a barcode field similar to those on supermarket products.

ii) Distributed database system

In a *distributed database system*, the actual database and the DBMS software are distributed from various sites that are connected by a computer network as shown in Figure 1.3.
iii) **Homogeneous distributed database systems**

Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily. For example, library information systems by the same vendor, such as Geac Computer Corporation, use the same DBMS software which allows easy data exchange between the various Geac library sites.

iv) **Heterogeneous distributed database systems**

In a heterogeneous distributed database system, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites. For example, the various library database systems use the same machine-readable cataloguing (MARC) format to support library record data exchange.

1.6.4 Licensing

This applies in two different ways - which is either a database management system is open-source or proprietary. Open-source means that computer software with its source code is made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. On the other hand, proprietary DBMSs creators and owners have all the rights to sell, modify or distribute.
1.7.5 Generic or Special Purpose

Generic DBMSs usage is not specific for any area of application i.e. they can be used in any application environment. However, special purpose databases are types of databases which specifically design for a special use.

Video lecture:

https://tinyurl.com/gm2j93g

Activity

Activity 1.0
Introduction to Databases Exercise

Motivation: To become conversant with basic features of selected DBMS.

Resources: Internet access and Unit 1 learning material.

What to do:
- List down five open-source DBMSs and their application environment.
- List down ten proprietary DBMSs and their application environment.
- List down at least three single-user DBMSs.

How to do it:
By using Internet search engine the student will have to attempt this activity.

Duration: Expect to spend about 1.30 hour on this activity

Feedback: This is a peer-reviewed activity where the learner should share their findings with other learners in class.
Unit summary

In this unit you learned about the difference between data and information, database management systems, their properties, and usage and application environment. Finally, you have also learned about classification of database management systems.

Review Questions

Information: These review questions are aimed at assisting the students in understanding of the first chapter of this course. Therefore, it is advised that they should be posted in a discussion forum where each student can contribute.

1. Give detailed classification of databases.
2. What is the purpose of managing information?
3. Discuss the uses of databases in a business environment.
4. Why would you choose a database system instead of simply storing data in operating system files? When would it make sense not to use a database system?
5. What are the duties of a DBA? Describe the major difference between DBA responsibilities and application programmer duties in database environment.
6. List four types of users for database system in university applications, banking industry and hospitality industry; giving details for each role.
7. What are the properties of a database?
8. Give five database application environments, detailing each use for that particular application.
9. Describe four features (functions) you would expect to find in a DBMS.
References and Further Reading


Download this book for free at http://open.bccampus.ca


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Database Usage and Environment
3. Activity1.0
4. Review questions
Unit 2

Database History

Introduction

Databases have existed for so many years as computer technology was progressing between storage, memory, processor and networks. Database technology in itself has evolved between different eras of navigational, relational and post relational models. This unit will give us a background of why database approach came into existence and the limitation of the file-based systems which were in use before the advent of databases.

Unit Outcomes

Upon completion of this unit you should be able to:

- Explain how database technology has evolved.
- Describe the limitations of file-based systems.
- Identify different application areas of database.

Terminologies

Data: Single fact or a piece of information.

File-based systems: An application program designed to manipulate data files.

DBMS: A powerful software tool that allows you to store, manipulate and retrieve data in a variety of ways.

Database approach: Allows the management of large amounts of organizational information.

Concurrency: The ability of the database to allow multiple users access to the same record without adversely affecting transaction processing.

Data inconsistency: A situation where various copies of the same data are conflicting.
Data isolation: A property that determines when and how changes made by one operation become visible to other concurrent users and systems.

2.1 Before the Advent of Databases

2.1.1 File-based Systems

One way to keep information on a computer is to store it in permanent files. A company system has a number of application programs and each of these is designed to manipulate data files. These application programs are written as per the request of the users in the organization. New applications are added to the system as the need arises. The system just described is called the file-based system.

Consider a traditional banking system that uses the file-based system to manage the organization’s data shown in Figure 2.1. As we can see, there are different departments in a bank. Each has its own applications that manage and manipulate different data files. For banking systems, the programs may be used to debit or credit an account, find the balance of an account, add a new mortgage loan and generate monthly statements.

Figure 2.1: Example of a file-based system used by banks to manage customers’ data, by A. Watt.
2.1.2 Limitations of the file-based system

Using the file-based system to keep organizational information has a number of disadvantages. Listed below are five examples.

i) Data redundancy

Often, within an organization, files and applications are created by different programmers from various departments over a long period of time. This can lead to data redundancy --, a situation that occurs in a database when a field needs to be updated in more than one table. This practice can lead to several problems such as:

   • Inconsistency in data format

The same information being kept in several different places (files) Data inconsistency, a situation where various copies of the same data are conflicting, wastes storage space and duplicates effort

ii) Data isolation

Data isolation is a property that determines when and how changes made by one operation become visible to other concurrent users and systems. This issue occurs in a concurrency situation. This is a problem because it is difficult for new applications to retrieve the appropriate data, which might be stored in various files.

iii) Integrity problems

A problem with data integrity is another limitation of using a file-based system. It refers to the maintenance and assurance that the data in a database are correct and consistent. Factors to consider when addressing this issue are:

   • Data values must satisfy certain consistency constraints that are specified in the application programs.
   • It is difficult to make changes to the application programs in order to enforce new constraints.

iv) Security problems

Security can be a problem with a file-based approach because:

   • There are constraints regarding accessing privileges.
   • Application requirements are added to the system in an ad-hoc manner so it is difficult to enforce constraints.
v) Concurrency access
Concurrency is the ability of the database to allow multiple users access to the same record without adversely affecting transaction processing. A file-based system must manage, or prevent, concurrency by the application programs. Typically, in a file-based system, when an application opens a file, that file is locked. This means that no one else has access to the file at the same time. In database systems, concurrency is managed thus allowing multiple users access to the same record. This is an important difference between database and file-based systems.

2.2 Database Approach
The difficulties that arise from using the file-based system have prompted the development of a new approach in managing large amounts of organizational information called the database approach. Databases and database technology play an important role in most areas where computers are used, including business, education and medicine. To understand the fundamentals of database systems, we will start by introducing some basic concepts in this area.

2.2.1 Role of Database in Business
Everyone uses a database in some way, even if it is just to store information about their friends and family. That data might be written down or stored in a computer by using a word-processing program or it could be saved in a spreadsheet. However, the best way to store data is by using database management software. This is a powerful software tool that allows you to store, manipulate and retrieve data in different ways. Most companies keep track of customer information by storing it in a database. This data may include customers, employees, products, orders or anything else that assists the business with its operations.

2.2.2 Data
Data are factual information such as measurements or statistics about objects and concepts. We use data for discussions or as part of a calculation. Data can be a person, a place, an event, an action or any one
of a number of things. A single fact is an element of data, or a *data element*.

If data are information and information is what we are in the business of working with, you can start to see where you might be storing it. Data can be stored in:

- Filing cabinets
- Spreadsheets
- Folders
- Ledgers
- Lists
- Piles of papers on your desk

All of these items store information, and so too does a database. Because of the mechanical nature of databases, they have terrific power to manage and process the information they hold. This can make the information they house much more useful for your work.
Activity

Activity 2.0
Data History Activity

Motivation: To differentiate the use of database application and spreadsheets.

Resources: Unit 2 learning materials, selected DBMS software and spreadsheet software.

What to do: Answer the following questions:

1. List down and describe each characteristic of the database approach.
2. What are the five major parts of a database system?
3. What is data redundancy, and which characteristics of the file system can lead to it?
4. What are some disadvantages of database systems?
5. List some of the functions of a DBMS.
6. Why is a spreadsheet NOT a database?

Duration: Expect to spend about 2 hours on this activity.

Feedback: The learner should submit this activity to the course instructor for assessment and feedback.

Unit summary

In this unit you learned about the evolution of database technology -- starting from the usage of file-based systems and the eventual advent of databases. This unit also, gave an overview of meaning of data and the role of databases in a business.
Review Questions

**Aim:** The review questions on this unit are aimed at determining the student’s knowledge on concepts covered in unit 2, database History. Discussion forum is highly recommended for conducting these review questions where each student can contribute.

1. Compare and contrast between data and information.
2. Explain the differences between file and record.
3. Explain the role and significance of databases in any society.
4. Why is it important to avoid data redundancy?
5. Select and briefly explain three disadvantages of the file based approach that is resolved by the database approach.
6. Explain the difference between data and information.
7. One of the advantages of database systems over file-based systems is said to be the concurrency control. Explain this term and why it is an advantage.
References and Further Reading:


Download this book for free at http://open.bccampus.ca


Download this book for free at http://open.bccampus.ca


Download this book for free at http://open.bccampus.ca
Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book *Database Design* by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Database Usage and Environment
3. Activity2.0
4. Summary
5. Review questions
Unit 3

Data Modelling

Introduction

Database Architecture plays a significant role in ensuring proper creation and management of a database. Data which is stored in a database can be seen from different levels, which we will be introduced to, in this unit and learn how to apply them in order to describe the structure of the database.

Unit Outcomes

Upon completion of this unit you will be able to:

- Explain the basic concepts of data modelling.
- Identify different levels of database architecture as viewed by users.
- Explain the concept of data independence and its importance in a database system
- Compare and contrast between logical and physical data independence.

Terminologies

Conceptual Model: The logical structure of the entire database.

Data independence: The immunity of user applications to changes made in the definition and organization of data in application program designed to manipulate data files.

Data model: A collection of concepts or notations for describing data, data relationships, data semantics and data constraints cords are organised in levels.

Database logical design: Defines a database in a data model of a specific DBMS the first step in the process of database design.

Database physical design: Defines the internal database storage structure, file organization and indexing techniques.

Physical Data The immunity of the internal model to changes in
Logical Data Independence: The ability to change the logical schema without changing the external schema.

3.1 Data Modelling

3.1.1 Introduction to Data Modelling

*Data modeling* is the first step in the process of database design. This step is sometimes considered to be a high-level and abstract design phase, also referred to as conceptual design. The aim of this phase is to describe:

- The data contained in the database (e.g., entities: students, lecturers, courses, subjects)
- The relationships between data items (e.g., students are supervised by lecturers; lecturers teach courses)
- The constraints on data (e.g., student number has exactly eight digits; a subject has four or six units of credit only)

In the second step, the data items, the relationships and the constraints are all expressed using the concepts provided by the high-level data model. Because these concepts do not include the implementation details, the result of the data modeling process is a (semi) formal representation of the database structure. This result is quite easy to understand so it is used as reference to make sure that all the user’s requirements are met.

The third step is database design. During this step, we might have two sub-steps: one called *database logical design* which defines a database in a data model of a specific DBMS, and another is called *database physical design* which defines the internal database storage structure, file organization or indexing techniques. These two sub-steps are database implementation and operations/user interfaces building steps.

In the database design phases, data are represented using a certain data model. The *data model* is a collection of concepts or notations for describing data, data relationships, and data semantics and data
constraints. Most data models also include a set of basic operations for manipulating data in the database.

### 3.1.2 Degree of Data Abstraction

In this section, we will look at the database design process in terms of specificity. Database design starts at a high level and proceeds to an ever-increasing level of detail. For example, when building a home, you start with how many bedrooms and bathrooms the home will contain -- whether it will be on one level or multiple levels, etc. The next step is to get an architect to design the home from a more structured perspective. This level gets more detailed with respect to actual room sizes, how the home will be wired, where the plumbing fixtures will be placed, etc. The last step is to hire a contractor to build the home. That’s looking at the design from a high level of abstraction to an increasing level of detail.

The database design is very much like that. It starts with users identifying the business rules; then the database designers and analysts create the database design; and then the database administrator implements the design using a DBMS.

### 3.2 Database Architecture Levels (Views)

As it was detailed above, database architecture provides a base for the database to have capability, reliability, effectiveness and efficiency in meeting user requirements. In addition, a correct database architecture should provide data independence and easy of design. In order for the design to meet these requirements, sophisticated database architecture is used, providing a number of levels of data abstraction or data definition. The levels (view) are listed as follows:

#### 3.2.1 External Level

External level or model represents the end user’s view of the database. It contains multiple different external views where each view represents a specific business unit of an organisation. This is a view which is closely related to the real world as perceived by each user. This model is aimed at facilitating application program development and assisting database designer in identifying data for each user requirement.
3.2.2 **Conceptual Level**

Conceptual level provides flexible data-structuring capabilities and a "community view": the logical structure of the entire database. This level has the following details:

i. Contains data stored in the database

ii. Shows relationships among data including:
   - Constraints
   - Semantic information (e.g., business rules)
   - Security and integrity information

iii. Considers a database as a collection of entities (objects) of various kinds.

iv. Are the basis for identification and high-level description of main data objects; they avoid details

v. Are database independent regardless of the database you will be using.

3.2.3 **Internal Level**

The Internal level adapts the conceptual level design to a specific DBMS which makes it software-dependent. The three best-known examples of internal view are the relational data model, the network data model and the hierarchical data model. This internal level:

i. Considers a database as a collection of fixed-size records

ii. Are closer to the physical level or file structure

iii. Are representations of the database as seen by the DBMS.

iv. Requires the designer to match the conceptual model’s characteristics and constraints to those of the selected implementation model.

v. Involve mapping the entities in the conceptual model to the tables in the relational model.
3.2.4 Physical Level

Physical level is the physical representation of the database and the lowest level of abstractions. Physical level is specifically based on:

i. How the data is stored. It deals with:
   - Run-time performance
   - Storage utilization and compression
   - File organization and access methods
   - Data encryption

ii. The physical level is managed by the operating system (OS)

iii. Provide concepts that describe the details of how data are stored in the computer’s memory.

iv. Software and hardware dependent

3.3 Data Abstraction Layer

In a pictorial view, you can see how the different models work together. Let’s look at this from the highest level -- i.e. the external model.

The external model is the end user’s view of the data. Typically a database is an enterprise system that serves the needs of multiple departments. However, one department is not interested in seeing other departments’ data (e.g., the human resources (HR) department does not care to view the sales department’s data). Therefore, one user’s view will differ from another user.

The external model requires that the designer subdivide a set of requirements and constraints into functional modules that can be examined within the framework of their external models (e.g., human resources versus sales).

As a data designer, you need to understand all the data so that you can build an enterprise-wide database. Based on the needs of various departments, the conceptual model is the first model created.
At this stage, the conceptual model is independent of both software and hardware. It does not depend on the DBMS software used to implement the model. It does not depend on the hardware used in the implementation of the model. Changes in either hardware or DBMS software have no effect on the database design at the conceptual level.

Once a DBMS is selected, you can then implement it. This is the internal model. Here you create all the tables, constraints, keys, rules, etc. This is often referred to as the logical design. The physical model is simply the way the data is stored on disk. Each database vendor has its own way of storing the data. Figure 3.1 illustrates the data abstraction layers.

![Data abstraction layers](image)

Figure 3.1: Data abstraction layers, by A. Watt

### 3.4 Schema

A *schema* is an overall description of a database, and it is usually represented by the *entity relationship diagram (ERD)*. There are many sub-schemas that represent external models and thus display external views of the data. Below is a list of items to consider during the design process of a database.

i. External schemas: there are multiples of these.

ii. Multiple sub-schemas: these display multiple external views of the data.

iii. Conceptual schema: there is only one. These schemas include data items, relationships and constrain one of this.
3.5 Data Independence

Data independence refers to the immunity of user applications to changes made in the definition and organization of data. Data abstractions expose only those items that are important or pertinent to the user. Complexity is hidden from the database user. Data independence and operation independence together form -- logical and physical data independence.

3.4.1 Logical data independence

A logical schema is a conceptual design of the database done on paper or a whiteboard -- much like architectural drawings for a house. The ability to change the logical schema, without changing the external schema or user view, is called logical data independence. For example, the addition or removal of new entities, attributes or relationships to this conceptual schema should be possible without having to change existing external schemas or rewrite existing application programs. In other words, changes to the logical schema (e.g., alterations to the structure of the database like adding a column or other tables) should not affect the function of the application (external views).

3.4.2 Physical data independence

Physical data independence refers to the immunity of the internal model to changes in the physical model. The logical schema stays unchanged even though changes are made to file organization or storage structures, storage devices or indexing strategy.

Physical data independence deals with hiding the details of the storage structure from user applications. The applications should not be involved with these issues, since there is no difference in the operation carried out against the data.
Activity

Activity 3.0
Database Architecture and Data Languages Activity

**Aim:** To develop knowledge of database architecture.

**Resources:** Unit 3 learning materials and word processor software.

**What to do:**
- Draw the ‘three-tier’ database architecture. Discuss the functions of each layer, and define the necessary tools involved.
- Draw ‘client-server architecture’ which is commonly used to implement a database system.

**Duration:** Expect to spend about 1 hour on this activity

**Feedback:** The learner should submit this activity to course instructor for assessment and feedback.
In this unit you learned on the basic feature of database architecture. The content included data abstraction layers which included internal view, external view, physical view and conceptual view. In addition, you have also learned about the concept of data independence and the difference between logical and physical data independence.
**Review Questions**

**Aim:** The review questions are aimed at assessing student understanding and knowledge on the concepts covered in unit three of the course. These questions work better if posted in a discussion forum for all students to contribute and share knowledge.

1. Describe the purpose of a conceptual design.
2. How is a conceptual design different from a logical design?
3. Compare and contrast between external view and conceptual view.
4. Compare and contrast between internal view and a physical view.
5. Which level of the database architecture does the database administrator work with?
6. Which level of the database architecture does the application programmer work with?
7. What is logical data independence and why is it important?
8. What is physical data independence?
9. Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?
10. Describe in details the three-tier database architecture.
References and Further Reading


   Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Activity 3.0
3. Summary
4. Review questions
Data Models

Introduction

Data Models provide the logical structure of data and fundamentally determines the manner in which data can be stored, organized, and manipulated. Therefore, in this unit, we will have an opportunity of learning the overview of database models, their various categories and structure for each.

Unit Outcomes

Upon completion of this unit you should be able to:

- Explain the basic concepts behind data models.
- Categorize different types of data models.
- Describe the basic differences between relational, hierarchical, object-oriented and network database models.

Terminologies

- **Data Model**: Logical structure of data.
- **Hierarchical Model**: A data model which represents data as a hierarchical tree structure.
- **Instance**: A record within a table.
- **Network Model**: A data model which represents data as record types.
- **Relation**: Another term for a table.
4.1 Overview

4.1.1 What is a Database Model?
A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated.

4.1.2 Categories of Database models
Database model can be categorized as high-level conceptual data models and record-based data models.

i) High-level Conceptual Data Models
High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee’s name, address and birth-date. A relationship represents an association among entities -- for example, an employee works on many projects. A relationship exists between the employee and each project.

ii) Record-based Logical Data Models
Record-based logical data models provide concepts users can understand but are not too far from the way data is stored in the computer. Three well-known data models of this type are relational data models, network data models and hierarchical data models.

4.2 Types of Database Models
The following are the most common database used in the industry. Each database model has its own characteristics which defines their application environment.

4.2.1 Hierarchical database Model
A hierarchical database model is a data model in which the data is organized into a tree-like structure. The data is stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value. The entity type of a
A record defines which fields the record contains. A record in the hierarchical database model corresponds to a row (or tuple) in the relational database model and an entity type corresponds to a table (or Figure 4.1 illustrates hierarchical model for a ‘university’ database where there are parent and child entities showing the hierarchy structure of the model.

![Hierarchical Database Model](image)

**Figure 4.1: Example of a hierarchical model, by G. Mbwete (2016)**

### 4.2.2 Network Database Model

The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice. Figure 4.2 shows an example of a ‘chain store’ database in a form of network database model.
4.2.3 Relational Database Model

The relational model represents data as relations, or tables. This is the most common model used in the database industry today. A relational database is based on the relational model developed by E.F. Codd in 1970s.

4.2.4 Object Oriented Database Model

Object-oriented database model conceptualize a database as a collection of objects, or reusable software elements, with associated features and methods. Multimedia and hypertext database are one the features of
object-oriented model. This is also known as post-relational database model, since it incorporates tables, but isn’t limited to tables; but more features as well. Figure 4.4 illustrates an object ‘customer’ with its associated attributes.

Figure 4.4: Illustration of a object-oriented data model, by G. Mbwete

Activity

Activity 4.0
Database Model Exercises

Aim: Compare and contrast between different database models.

Resources: Unit 4 learning materials and selected DBMS software.

What to do: Answer the following questions:
1. Explain the basic features of relational database model.
2. Are there any emerging database models apart from relational databases? Justify your answer.
3. What is object-relational database model? How common is model being used in the IT industry today?

Duration: Expect to spend about 2 hours on this activity.
Feedback: The learner should submit this activity to the course instructor for assessment and feedback.
Unit summary

In this unit you learned about database models and their categories. In addition, you have also learned about different types of database models which have formed the basis of databases in use in the industry.
References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Activity 4.0
3. Summary
Unit 5

Relational Data Model

Introduction

The relational data model provides techniques of structuring how data can be perceived. Relational Data Model provides conceptual view of data which is not provided by the E-R model which usually represents the relations between elements of a database. The relational data model assumes entities as tables and allows operations to be performed on them. In this unit, the learner will learn how to map ER models into relations.

Unit Outcomes

Upon completion of this unit you should be able to:

- Describe the fundamental concepts and notations in relational data model.
- Explain the various types of relationships in the Relational Data Model.
- Describe the basic properties of a table.

Terminologies

Relation: Subset of the Cartesian product of a list of domains characterized by a name. Technical name for table or file.

SQL: Structured Query Language (SQL): the standard database access language.

Domain: The original sets of atomic values used to model data; a set of acceptable values that a column is allowed to contain. Mechanism the system provides to maintain primary keys.

Atomic value: Each value in the domain is indivisible as far as the relational model is concerned.

Table: The same as relation or an entity or instance.
5.1 Overview

The relational data model was introduced by C. F. Codd in 1970. Currently, it is the most widely used data model. The relational model has provided the basis for:

i. Research on the theory of data/relationship/constraint
ii. Numerous database design methodologies
iii. The standard database access language called *structured query language* (SQL)
iv. Almost all modern commercial database management systems.
v. The relational data model describes the world as “a collection of inter-related relations (or tables).”

5.2 Fundamental Concepts in Relational Model

5.2.1 Relation/Table

A *relation*, also known as a *table* or *file*, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A *row*, or record, is also known as a *tuple*. The column in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.

The steps below outline the logic between a relation and its domains.

i. Given *n* domains are denoted by D1, D2, … Dn
ii. And *r* is a relation defined on these domains
iii. Then *r* ⊆ D1×D2×…×Dn

A database is composed of multiple tables and each table holds the data. Table 5.1 illustrates shows a relation known as ‘Course’ which contains details of a university ‘course’.
Table 5.1: ‘Course’ relation

<table>
<thead>
<tr>
<th>CourseCode</th>
<th>CourseName</th>
<th>CourseInstructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>Miss Grace</td>
</tr>
<tr>
<td>OIT216</td>
<td>System Security</td>
<td>MrRaitton</td>
</tr>
<tr>
<td>OIT218</td>
<td>Web Programming</td>
<td>MrOmbeni</td>
</tr>
</tbody>
</table>

5.2.2 Column/Field/Attribute

A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization. The principal storage units are called columns or fields or attributes. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.

Table 5.2 illustrates the attributes (characteristics) for ‘Course’ relation/table. This relation has four columns/field/attributes namely: Course_Code, Course_Name, Course_Instructor and Course_Unit which give characteristics of a ‘course’ table.

Table 5.2: Illustration of columns of a ‘Course’ table

<table>
<thead>
<tr>
<th>Course_CODE</th>
<th>Course_Name</th>
<th>Course_Instructor</th>
<th>Course_Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>Miss Grace</td>
<td>2</td>
</tr>
<tr>
<td>OIT216</td>
<td>System Security</td>
<td>MrRaitton</td>
<td>1</td>
</tr>
<tr>
<td>OIT218</td>
<td>Web Design</td>
<td>MrOmbeni</td>
<td>3</td>
</tr>
</tbody>
</table>
5.2.3 Domain

A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. The following are examples:

i. The domain of date of birth has a set of numbers in any date format.

ii. The domain of Marital Status has a set of possibilities: Married, Single, Divorced.

iii. The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.

iv. The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.

v. The domain of First Name is the set of character strings that represents names of people.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column.

5.2.4 Records/Row/Tuples

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form. Records allow us to do this. *Records* contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project. Table 5.3 below shows ‘Course’ relation which ‘five’ tuples/records.
Table 5.3: Illustration of rows/tuples of ‘Course’ relation

<table>
<thead>
<tr>
<th>Course_Code</th>
<th>Course_Name</th>
<th>Course_Instructor</th>
<th>Course_Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>MsGrace Mbwete</td>
<td>2</td>
</tr>
<tr>
<td>OIT216</td>
<td>System Security</td>
<td>MrEliahLukwaro</td>
<td>1</td>
</tr>
<tr>
<td>OIT218</td>
<td>Web Programming</td>
<td>MrOmbeni Mathias</td>
<td>3</td>
</tr>
<tr>
<td>OIT313</td>
<td>Network</td>
<td>Mrs Lilian Charles</td>
<td>4</td>
</tr>
<tr>
<td>OIT314</td>
<td>Programming</td>
<td>Mr Said Ally</td>
<td>2</td>
</tr>
</tbody>
</table>

5.2.5 Degree

The degree is the number of attributes/columns/fields in a table or relation. In the example below in table 5.4, for a ‘Course’ table, the degree is 4.

Table 5.4: Illustration of degree of a table – ‘Course table’

<table>
<thead>
<tr>
<th>Course_Code</th>
<th>Course_Name</th>
<th>Course_Instructor</th>
<th>Course_Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>Miss Grace Mbwete</td>
<td>2</td>
</tr>
<tr>
<td>OIT216</td>
<td>System Security</td>
<td>MrEliahLukwaro</td>
<td>1</td>
</tr>
<tr>
<td>OIT218</td>
<td>Web Programming</td>
<td>MrOmbeni Mathias</td>
<td>3</td>
</tr>
<tr>
<td>OIT313</td>
<td>Network</td>
<td>Mrs Lilian Charles</td>
<td>4</td>
</tr>
<tr>
<td>OIT314</td>
<td>Programming</td>
<td>Mr Said Ally</td>
<td>2</td>
</tr>
</tbody>
</table>
5.3 Properties of a Table

Tables in a relational data model have the following major characteristics:

i. A Table has a name that is distinct from all other tables in the database.

ii. There are no duplicate rows, each row is distinct.

iii. Entries in columns are atomic (no repeating groups or multivalued attributes)

iv. Entries from columns are from the same domain based on their data type:
   - Number (numeric, integer, float, smallint,…)
   - Character (string)
   - Date
   - Logical (true or false)

v. Operations combining different data types are disallowed.

vi. Each attribute has a unique/distinct name.

vii. The order of columns in a table is not important.

viii. The order of rows is not important.

Video lecture:

https://tinyurl.com/jxqzavf
Activity

Activity 5.0
Relational Database Model Exercises

Aim: To become conversant with basic Relational Database Model.

Resources: Unit 5 learning materials.

What to do: Study table 5.5 below and answer the following questions:
1. Using correct terminology, identify and describe all the components in Table 5.5.
2. What is the possible domain for field Staff_Office?
3. How many fields are shown? What is the degree of the table?
4. How many tuples does the table have?

Table 5.5: Staff Table

<table>
<thead>
<tr>
<th>Staff_ID</th>
<th>Staff_FirstName</th>
<th>Staff_LastName</th>
<th>Staff_Office</th>
<th>Staff_Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>608</td>
<td>Grace</td>
<td>Mbwete</td>
<td>ODL-908</td>
<td>ICT</td>
</tr>
<tr>
<td>702</td>
<td>Elia</td>
<td>Ahidi</td>
<td>Block E-ICT</td>
<td>ICT</td>
</tr>
<tr>
<td>1002</td>
<td>Lillian</td>
<td>Charles</td>
<td>ODL-806</td>
<td>Nutrition</td>
</tr>
<tr>
<td>2006</td>
<td>Cathy</td>
<td>Gerald</td>
<td>Block C-202</td>
<td>P/Science</td>
</tr>
<tr>
<td>3117</td>
<td>Martin</td>
<td>Clemence</td>
<td>Bungo-101</td>
<td>Nutrition</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 2 hours on this activity.

Feedback: This activity should be submitted to the course instructor for assessment and feedback.
Unit summary

In this unit you have learnt about Relational Data Model which was developed by C.F. Codd in the 1970s. The unit content had covered fundamentals concepts behind Relational Database Model which includes details on the relations/tables and its properties (fields/columns, tuples/rows, degree and domain).
Review Questions

Motivation: The review questions are aimed at assessing student understanding and knowledge on the concepts covered in unit five of the course. These questions work better if posted in a discussion forum for all students to contribute and share knowledge.

1. Define the term relational database schema (the schema of a whole relational database).
2. Define the term relation schema key. List the properties of a relation schema key.
3. What is the difference between a relation and a relation schema, and what is their relationship?

With examples, please explain the following terms:

- Relation
- Domain
- Attribute
- Attribute domain
- Relation instance
- Relation cardinality
- Relation degree
References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Activity4.0
3. Summary
Unit 6

Entity Relationship Model

Introduction

For a correct design and implementation of database, there is a need to understand which entities should hold data and identify the connections that may exist between entities. In this unit, we are going to learn about the Entity-Relationship Model which provides a technique in creating a graphical view of the different elements of a database as well as the relationships between them. In addition, we are also going to learn the drawing conventions of the E-R model, beginning with those conventions used to represent a single entity, and concluding with conventions used to represent all relations in a database.

Unit Outcomes

Upon completion of this unit you will be able to:

- Describe the significance ER Model in database design.
- Explain the entity-relationship model and its components.
- Convert user requirements into an ER Model.

Terminologies

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>A thing with distinct and independent existence.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Characteristic of an entity.</td>
</tr>
<tr>
<td>Relationship</td>
<td>Connection between entities/tables in a database.</td>
</tr>
<tr>
<td>ERD</td>
<td>Entity Relationship Diagram - also called an ER schema, are represented by ER diagrams. These are well suited to data modeling for use with databases.</td>
</tr>
<tr>
<td>Primary Key</td>
<td>A special relational database table column (or combination of columns) designated to uniquely</td>
</tr>
</tbody>
</table>
6.1 Introduction to ER Model

The entity relationship (ER) data model was developed for database design by Peter Chen and published in a 1976 paper. It is well suited to data modeling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams. ER modeling is based on two concepts:

- **Entities**, defined as tables that hold specific information (data).
- **Relationships**, defined as the associations or interactions between entities.

For the rest of this unit, we will use a sample database called the UNIVERSITY database to illustrate the concepts of the ER model. This database contains information about lecturers, students and courses. Important points to note include:

- There are must be lecturers in the university. Each lecturer has a unique identification (PF Number), a name, location of the office, salary, birthdate, start date and contacts.
- A department in the department controls a number of academic programs, each of which has a unique name, a unique number, lecturers and a budget.
- Each academic program has several courses which have course identification (course code), lecturers, units and subjects.
- Students are has to enroll in an academic program and register for its respective in the university. The student has student ID, name and contacts.
6.2 Entities

6.2.1 Definition of an Entity
An entity is an object in the real world with an independent existence and can be differentiated from other objects. An entity might be

- An object with physical existence, e.g. a lecturer or a student.
- An object with conceptual existence, e.g. a course and a job.

An Entity Type defines a collection of similar entities.

An Entity Set is a collection of entities of an entity type at a point of time.

In ER diagrams, an entity type is represented by a name in a box.

![Student Entity](image)

Figure 6.1: A student entity, by G. Mbwete

6.2.2 Classification of Entities
Entities are classified as strong entities, weak entities and existence dependency as detailed in sub-sections below.

i) Strong Entities
An entity is considered a ‘strong’ entity, if it has the following properties:

- If it can exist apart from all of its related entities.
- Kernels are strong entities.
- A table without a foreign key is or a table that contains a foreign key which can contain NULLS is a strong entity.

ii) Weak Entities
An entity is considered a ‘weak’ entity, if it has the following properties:

- These tables are existence dependent.
- They cannot exist without entity with which it has a relationship.
- Primary key is derived from the primary key of the parent entity.

For example; considering the lecturer’s spouse table in the UNIVERSITY database referred above. This is a weak entity because its PK is
dependent on the lecturer’s table whereby without a corresponding lecturer’s record, the spouse record could not exist.

6.2.3 Types of Entities
i) Independent Entities
Independent entities, also referred to as Kernels, are the backbone of the database. It is what other tables are based on. Kernels have the following characteristics:
- They are the ‘building blocks’ of a database
- The primary key may be simple or composite
- The primary key is not a foreign key
They do not depend on another entity for their existence. For example, Lecturer’s table and Course table in a UNIVERSITY database referred above.

ii) Dependent Entities
Dependent entities are used to connect two kernels together. Dependent entities have the following characteristics:
- They are said to be existent dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.

There are three options for the primary key for dependent entities:
- Use a composite of foreign keys of associated tables if unique
- Use a composite of foreign keys and qualifying column
- Create a new simple primary key

ii) Characteristic Entities
Characteristic entities provide more information about another table. These entities have the following characteristics:
- They represent multi-valued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
Options for primary key for this entity are either of the following:
- Foreign key plus a qualifying column
- Create a new simple primary key
  - Lecturer (PFNo, Name, Address, Age, Salary)
  - LecturerPhone(PFNo, Phone)

### 6.2.4 Attributes
Each entity is described by a set of attributes. E.g. Lecturer = (ID_No, Name, Address, Age, Salary).
Each attribute has a name, associated with an entity and is associated with a domain of legal values. However the information about attribute domain is not presented on the ER diagram.
In the diagram, each attribute is represented by an oval with a name inside.

![ER Diagram of Lecturer Entity](image)

**Figure 6.2: Illustration of attributes of 'lecturer' entity, by G. Mbwete**

#### i) Types of Attributes
There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.
- **Simple Attributes**
  Simple attributes are those drawn from the atomic value domains; they are also called single-valued attributes. In the UNIVERSITY database, an example of this would be: ID_No = {908}; Name = {Grace}; Age =
- **Composite Attribute**

  Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown in Figure 6.4, Address may consist of Number, Street and Suburb. So this would be written as follows:-

  \[
  \text{Address} = \{30 + \text{‘Tumaini’} + \text{‘Mikocheni’}\}
  \]
• **Multivalued attributes**

Multivalued attributes are attributes that have a set of values for each entity. Examples of a multivalued attribute from the UNIVERSITY database, as seen in Figure 6.5 below, are the degrees of a lecturer: BSc, MIT and PhD.

![Figure 6.5: Illustration of multi-valued attribute ‘degree’ of ‘lecturer’, by G. Mbwete](image)

• **Derived attributes**

Derived attributes are attributes that contain values calculated from other attributes. An example of this can be seen in Figure 6.6. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.

![Figure 6.6: Illustration of derived attribute ‘Age’ of ‘lecturer’ entity, by G. Mbwete](image)
6.2.5 Keys

An important constraint on an entity is the key. The key is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set. The following are types of keys:

i) Candidate Key

A candidate key is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

From our UNIVERSITY database example, if the entity is Lecturer (LID, FirstName, LastName, SIN, Address, Phone, BirthDate, Salary, DepartmentID), possible candidate keys are:

- LID, SIN
- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and DepartmentID – assuming two people with the same last name don’t work in the same department

ii) Composite key

A composite key is composed of two or more attributes, but it must be minimal. Using the example from the candidate key section, possible composite keys are:

- First Name and Last Name – assuming there is no one else in the company with the same name.
- Last Name and Department ID – assuming two people with the same last name don’t work in the same department.

iii) Primary Key

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
• A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.

In the following example, EID is the primary key:
Lecturer (LID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

iv) Secondary key
A secondary key is an attribute used strictly for retrieval purposes (can be composite), for example: Phone and Last Name.

v) Alternate key
Alternate keys are all candidate keys not chosen as the primary key.

vi) Foreign key
A foreign key (FK) is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type. In the UNIVERSITY database example below, DepartmentID is the foreign key:
Lecturer (LID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)
Department (DepartmentID, DepartmentName, DepartmentSpecialities)

6.3 Relationships

6.3.1 Introduction
Relationships are the glue that holds the tables together. They are used to connect related information between tables. Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. For example:

• Customer(CustID, CustName)
• Order(OrderID, CustID, Date)
A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

- Course(CourseCode, DepartmentID, Description)
- Class(CrsCode, Section, ClassTime…)

6.3.2 Types of Relationships

Below are descriptions of the various types of relationships.

i) One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, in a UNIVERSITY database; one department has many lecturers. Figure 6.7 shows the relationship (one to many) of one of these lecturers to the department.

![Figure 6.7: Example of a one to many relationship, by G.Mbwete](image)

ii) One-to-One Relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.
An example from the UNIVERSITY database is one lecturer is associated with one spouse, and one spouse is associated with one lecturer.

iii) **Many-to-many relationship**

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.
- The composite entity table must contain at least the primary keys of the original tables.
- The linking table contains multiple occurrences of the foreign key values.
- Additional attributes may be assigned as needed.
- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

Figure 6.8 shows another aspect of the M:N relationship where a lecturer has different start dates for different academic programs. Therefore, we need a JOIN table that contains the LID, PrgrmCode and StartDate.
Example of mapping an M:N binary relationship type

- For each M:N binary relationship, identify two relations.
- A and B represent two entity types participating in R.
- Create a new relation S to represent R.
- S needs to contain the PKs of A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table R can be the PK.
- The combination of the primary keys (A and B) will make the primary key of S.

iv) Unary relationship (recursive)

A unary relationship, also called recursive, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See Figure 6.8 for an example. For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.
v) Ternary Relationship

A ternary relationship is a relationship type that involves many to many relationships between three tables. The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side. In most cases of an n-ary relationship, all the participating entities hold a many side.

Video lecture:

https://tinyurl.com/h22xdl
https://tinyurl.com/htfkunh

https://tinyurl.com/jbbrq9g
## Activity

<table>
<thead>
<tr>
<th>Activity 6.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER Modelling Exercises</td>
</tr>
</tbody>
</table>

**Motivation:** To become conversant with basic ER Modelling techniques.

**Resources:** Unit 5 and 6 learning materials and ER Diagram Tool.

**What to do:**
Read the following scenario carefully.
A manufacturing company produces products. Product information stored is product name, id, and quantity on hand. These products are made up of many components. Each component can be supplied by one or more suppliers. Component information kept is component id, name, description, suppliers who supply them, and which products they are used in. Create an ERD to show how you would track this information. Show entity names, primary keys, attributes for each entity, relationships between the entities and cardinality.

**How to do it:**
For the above scenario, draw an entity-relationship model diagram that captures the given requirements. Once done, validate the model against the requirements to make sure nothing was missed. Remember that the four entity-relationship elements we want to pull out of these requirements are: entities, attributes, identifiers, and relationships.

**Duration:** Expect to spend about 2 hour on this activity.

**Feedback:** The learner should submit this activity for assessment to the course instructor and get a feedback on this activity.
Activity 6.1
ER Modelling Exercises

Motivation: To become conversant with basic ER Modelling techniques.

Resources: Unit 5 and 6 learning materials and ER Diagram Tool.

What to do:
Read the following scenario carefully;

The SSR marketing department needs a database to keep track of its product line. The department creates new and unique razor models all of the time (e.g. The Ripper, Slashmatic, Yankie 3000, Lady Yankie, ElectrothrobYankie Deluxe). Each model features a matched pair of a cartridge and handle. A model may have alternate handles (e.g. battery powered vibration, metallic blue finish, Hello Kitty™ decal) but only uses one cartridge type. To inflate costs, neither handles nor cartridges are reused for new models. Part numbers for the various handles and cartridges are recorded. The company packages together sets of two cartridges and a handle under the model name. The company also packages sets of eight replacement cartridges under the model name. The retail price of each package is recorded.

How to do it:
For the above scenario, draw an entity-relationship model diagram that captures the given requirements. Once done, validate the model against the requirements to make sure nothing was missed. Remember that the four entity-relationship elements we want to pull out of these requirements are: entities, attributes, identifiers, and relationships.

Duration: Expect to spend about 2 hour on this activity.

Feedback: The learner should submit this activity for assessment to the course instructor and get a feedback on this activity.
Unit summary

In this unit you learned about Entity relationship Model which helps in the logical design stage of a relational database. The unit content has given details on entities and its types, attributes and its characteristics and finally; relationships which are the basis/connectivity between entities/tables in the database.
Case Study

**Aim:** This section contains several case studies where they are aimed at motivating the learner to develop techniques of converting user requirements into an ER Diagram for each case.

**Duration:** Expect to spend about 1.30 hour on this activity.

**Feedback:** These case studies should be submitted to the course instructor for assessment and feedback.

**Case Study I**
UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system. Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail center. Retail centers are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g., flight, truck), and a deliveryRoute. Please create an Entity Relationship diagram that captures this information about the UPS system. Be certain to indicate identifiers and cardinality constraints.

**Case Study II**
A car dealership sells both new and used cars, and it operates a service facility. Base your design on the following business rules:
- A salesperson may sell many cars, but each car is sold by only one salesperson.
- A customer may buy many cars, but each car is sold to only one customer.
- A salesperson writes a single invoice for each car he or she buys.
- A customer gets an invoice for each car he or she buys.
- A customer may come in just to have his or her car serviced, that is, one need not buy a car to be classified as a customer.
- When a customer takes one or more cars in for repair or service, one service ticket is written for each car.
- The car dealership maintains a service history for each of the cars serviced. The service records are referenced by the car’s serial number.
- A car brought in for service can be worked on by many mechanics, and each mechanic may work on many cars.
- A car that is serviced may or may not need parts. (For example, adjusting a carburetor or cleaning a fuel injector nozzle does not
require the use of parts).

Case Study III
Discount Diva Prop Rentals
This company rents WWII-vintage propeller aircraft exclusively to celebrity female singers.

• The company caters exclusively to true divas.
• Some divas have pilot licenses, and only those may rent aircraft to fly between European capital city airports.
• Depending on an aircraft model's seating capacity, more than one diva can join in on the trip.
• Aircraft are identified by their tail number.
• Because of the fragile nature of the propeller aircraft, a single mechanic is assigned to each. New aircraft may not yet have an assigned mechanic, and may not be rented until they do.
• An aircraft can be used for multiple rentals, but not simultaneously. To make sure an aircraft isn't overbooked, each rental records both the departure and arrival dates.
• A rental starts at one airport (departure) and ends at a second airport (arrival). The arrival and departure airports may be the same (a round-trip).
• When more than one diva share a rental, the seating position of each diva is recorded.

Case Study IV
Secretive Sci-Fi Star Maps

• This company creates and sells maps that indicate the address of select science fiction film stars and also movie shoot locations. Several maps are needed to cover the large Los Angeles region. The map borders don't overlap one other. Each sci-fi star's address is shown on the appropriate map at the proper coordinates. The film or films that each star has appeared in are recorded, along with the role played.
• Customers of SSFSM often want to find all of the stars that appeared in a certain movie (e.g. Star Trek II, Star Wars IV, The Adventures of Buckaroo Banzai Across the 8th Dimension).
• Special shoot locations for some sci-fi films are also displayed on the maps.

Case Study V
Raining Cats 'n Dogs Animal Shelter
The shelter is pretty selective in which animals it takes in, and also what it chooses to record for the two species.

• The animal shelter accepts just cats and dogs.
• For both cats and dogs, the name and gender of the animal is recorded, as well as whether it has been neutered or not.
• For cats, the mother of the cat is noted, if the mother is known to the shelter. For any given mother, the shelter wants to be able to find all descendants through potentially many generations.
• Dogs are classified by breed. Dogs can be of one breed (purebred), two breeds (crossbreed), or just be a mutt. The specific breeds, except in the case of a mutt, are tracked.
• The names, addresses, and telephone numbers of people adopting a dog or a cat (adopters) is recorded.
• The names, addresses, and telephone numbers of people surrendering a dog or a cat (donators) is recorded.
• For both adopters and donators, a reference to the dog or cat they adopted or donated, respectively, is kept.
• Adopters can adopt many dogs or cats, or may just be interested in adoption.

Case Study VI
Draw an EER diagram of the conceptual schema for another part of a University database, described as follows:

• Academic staff, general staff and students are the only persons at the university.
• Each person is either an academic staff, or a general staff, or a student.
• A person is uniquely identified by a PerId (person's ID), and has a Name, and an Address. An Address is composed of HouseNo, Street, and City.
• A characteristic property of a student is that she/he has at least one Major and one NoOfPts (number of points) for each major.
• An academic staff has a Position and an AcQual (academic qualification).
• A general staff has a GenPos (general position).
• An academic staff teaches at most one course, whereas a student takes at least one course.
• A course is uniquely identified by a CourId (course ID), and has a CourName (course name).
• Each course is taught by at least one academic staff, and can be taken by many students, but there may be courses that are not taken by any students.
• Each course can use more than one textbook, but there may be courses with no textbook.
• A textbook is uniquely identified by the course which uses the book, and by an OrdNo. The attribute OrdNo is the ordinal number of the book in the list of the textbooks of a particular course. A book also has a Title.
References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The ER Case Studies are authored by David Rogers from Yukon College.

The following material was written by Grace Mbwete:

1. Introduction
2. Summary
Unit 7

Integrity Rules and Constraints

Introduction

One of the important functionality of a DBMS is to enable the specification of integrity constraints and to enforce them. Constraints are useful because they allow a designer to specify the semantics of data in the database. Constraints are the rules that force DBMSs to check that data satisfies the semantics. The concepts which will be covered in this unit, will assist the student in understanding and be able define the database constraints during the practical sessions of this course.

Unit Outcomes

Upon completion of this unit you will be able to:

- Describe the basic concepts of data integrity rules.
- Specify integrity constraints and how to enforce them.
- Identify business rules when gathering user requirements.

Terminologies

Constraints: The rules that force DBMSs to check that data satisfies the semantics.

Entity Integrity: Requires that every table have a primary key; neither the primary key, nor any part of it, can contain null values.

Integrity constraints: Logical statements that state what data values are or are not allowed and which format is suitable for an attribute.

Business rules: Obtained from users when gathering requirements and are used to determine cardinality.

Cardinality: Expresses the minimum and maximum number of entity occurrences associated with one occurrence of a related entity.
7.1 Overview

Constraints are a very important feature in a relational model. In fact, the relational model supports the well-defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database. *Constraints* are the rules that force DBMSs to check that data satisfies the semantics.

7.2 Integrity Constraints

Domain restricts the values of attributes in the relation and is a constraint of the relational model. However, there are real-world semantics for data that cannot be specified if used only with domain constraints. We need more specific ways to state what data values are or are not allowed and which format is suitable for an attribute. For example, the Employee ID (EID) must be unique or the employee Birthdate is in the range [Jan 1, 1950, Jan 1, 2000]. Such information is provided in logical statements called *integrity constraints*. There are several kinds of integrity constraints, as detailed below:

*7.2.1 Entity Integrity*

To ensure *entity integrity*, it is required that every table have a primary key. Neither the PK nor any part of it can contain null values. This is because null values for the primary key mean we cannot identify some rows. For example, in the EMPLOYEE table, Phone cannot be a primary key since some people may not have a telephone.

*7.2.2 Referential Integrity*

*Referential integrity* requires that a foreign key must have a matching primary key or it must be null. This constraint is specified between two tables (parent and child); it maintains the correspondence between rows
in these tables. It means the reference from a row in one table to another table must be valid.

Examples of referential integrity constraint in the Customer/Order database of the Company:

- Customer(CustID, CustName)
- Order(OrderID, CustID, OrderDate)

To ensure that there are no orphan records, we need to enforce referential integrity. An orphan record is one whose foreign key FK value is not found in the corresponding entity – the entity where the PK is located. Recall that a typical join is between a PK and FK.

The referential integrity constraint states that the customer ID (CustID) in the Order table must match a valid CustID in the Customer table. Most relational databases have declarative referential integrity. In other words, when the tables are created the referential integrity constraints are set up.

Here is another example from a Course/Class database:

- Course(CrsCode, DeptCode, Description)
- Class(CrsCode, Section, ClassTime)

The referential integrity constraint states that CrsCode in the Class table must match a valid CrsCode in the Course table. In this situation, it’s not enough that the CrsCode and Section in the Class table make up the PK, we must also enforce referential integrity.

When setting up referential integrity it is important that the PK and FK have the same data types and come from the same domain, otherwise the relational database management system (RDBMS) will not allow the join. RDBMS is a popular database system that is based on the relational model introduced by E. F. Codd of IBM’s San Jose Research Laboratory. Relational database systems are easier to use and understand than other database systems.

i) Referential integrity in Microsoft Access

In Microsoft (MS) Access, referential integrity is set up by joining the PK in the Customer table to the CustID in the Order table. See Figure 9.1 for
a view of how this is done on the Edit Relationships screen in MS Access.

![Edit Relationships screen in MS Access](image)

Figure 7.1: Referential access in MS Access, by A. Watt.

Referential integrity using Transact-SQL (MS SQL Server)
When using Transact-SQL, the referential integrity is set when creating the Order table with the FK. Listed below are the statements showing the FK in the Order table referencing the PK in the Customer table.

```sql
CREATE TABLE Customer
  ( CustID INTEGER PRIMARY KEY,
  CustName CHAR(35) )
CREATE TABLE Orders
  ( OrderID INTEGER PRIMARY KEY,
  CustID INTEGER REFERENCES Customer(CustID),
  OrderDate DATETIME )
```

ii) **Foreign key rules**
Additional foreign key rules may be added when setting referential integrity, such as what to do with the child rows (in the Orders table) when the record with the PK, part of the parent (Customer), is deleted or changed (updated). For example, the Edit Relationships window in MS Access (see Figure 9.1) shows two additional options for FK rules:
Cascade Update and Cascade Delete. If these are not selected, the system will prevent the deletion or update of PK values in the parent table (Customer table) if a child record exists. The child record is any record with a matching PK.

In some databases, an additional option exists when selecting the Delete option called Set to Null. In this is chosen, the PK row is deleted, but the FK in the child table is set to NULL. Though this creates an orphan row, it is acceptable.

7.2.3 Enterprise Constraints

Enterprise constraints – sometimes referred to as semantic constraints – are additional rules specified by users or database administrators and can be based on multiple tables. Here are some examples.

- A class can have a maximum of 30 students.
- A teacher can teach a maximum of four classes per semester.
- An employee cannot take part in more than five projects.
- The salary of an employee cannot exceed the salary of the employee’s manager.

7.3 Business Rules

*Business rules* are obtained from users when gathering requirements. The requirements-gathering process is very important, and its results should be verified by the user before the database design is built. If the business rules are incorrect, the design will be incorrect, and ultimately the application built will not function as expected by the users. Some examples of business rules are:

- A teacher can teach many students.
- A class can have a maximum of 35 students.
- A course can be taught many times, but by only one instructor.
- Not all teachers teach classes.

7.3.1 Cardinality and connectivity

Business rules are used to determine cardinality and connectivity. *Cardinality* describes the relationship between two data tables by expressing the minimum and maximum number of entity
occurrences associated with one occurrence of a related entity. In Figure 9.2, you can see that cardinality is represented by the innermost markings on the relationship symbol. In this figure, the cardinality is 0 (zero) on the right and 1 (one) on the left.

![Figure 7.2: Position of connectivity and cardinality on a relationship symbol, by A. Watt](image)

The outermost symbol of the relationship symbol, on the other hand, represents the connectivity between the two tables. **Connectivity** is the relationship between two tables, e.g., one to one or one to many. The only time it is zero is when the FK can be null. When it comes to participation, there are three options to the relationship between these entities: either 0 (zero), 1 (one) or many. In Figure 9.2, for example, the connectivity is 1 (one) on the outer, left-hand side of this line and many on the outer, right-hand side.

Figure 7.3 below, shows the symbol that represents a ‘one to many’ relationship.

![Figure 7.3: One to many relationship](image)

In Figure 7.4 both inner (representing cardinality) and outer (representing connectivity) markers are shown. The left side of this symbol is read as
minimum 1 and maximum 1. On the right side, it is read as: minimum 1 and maximum many.

Figure 7.4: Illustration of cardinality

7.4 Relationship Types

The line that connects two tables, in an ERD, indicates the relationship type between the tables: either identifying or non-identifying. An identifying relationship will have a solid line (where the PK contains the FK). A non-identifying relationship is indicated by a broken line and does not contain the FK in the PK. See the section in Chapter 8 that discusses weak and strong relationships for more explanation.

![Identifying and non-identifying relationship](image)

Figure 7.5: Identifying and non-identifying relationship, by A. Watt.

7.4.1 Optional relationships

In an optional relationship, the FK can be null or the parent table does not need to have a corresponding child table occurrence. The symbol, shown in Figure 7.6 below, illustrates one type with a zero and three prongs (indicating many) which is interpreted as zero OR many.
For example, if you look at the Order table on the right-hand side of Figure 9.7, you’ll notice that a customer doesn’t need to place an order to be a customer. In other words, the **many side** is optional.

The relationship symbol in Figure 7.7 above, can also be read as follows:

i. Left side: The order entity must contain a minimum of one related entity in the Customer table and a maximum of one related entity.

ii. Right side: A customer can place a minimum of zero orders or a maximum of many orders.

Figure 7.8, shows another type of optional relationship symbol with a zero and one, meaning zero OR one. The **one side** is optional.
7.4.3 **Mandatory relationships**

In a *mandatory relationship*, one entity occurrence requires a corresponding entity occurrence. The symbol for this relationship shows *one and only one* as shown in Figure 7.10. The one side is mandatory.
See Figure 7.11 as an example of how the one and only one mandatory symbol is used.

Figure 7.11: Example of a one and only one mandatory relationship symbol, by A. Watt.

Figure 7.12 illustrates what a one to many relationship symbol looks like where the many side is mandatory.

Figure 7.12: One to many with ‘many’ as mandatory, by A. Watt

Refer to Figure 7.13 for an example of how the one to many symbol may be used.
So far we have seen that the innermost side of a relationship symbol (on the left-side of the symbol in Figure 7.14) can have a 0 (zero) cardinality and a connectivity of many (shown on the right-side of the symbol in Figure 7.14), or one (not shown).

However, it cannot have a connectivity of 0 (zero), as displayed in Figure 7.15. The connectivity can only be 1.
The connectivity symbols show maximums. So if you think about it logically, if the connectivity symbol on the left side shows 0 (zero), then there would be no connection between the tables. The way to read a relationship symbol, such as the one in Figure 9.16, is as follows.

- The CustID in the Order table must also be found in the Customer table a minimum of 0 and a maximum of 1 times.
- The 0 means that the CustID in the Order table may be null.
- The left-most 1 (right before the 0 representing connectivity) says that if there is a CustID in the Order table, it can only be in the Customer table once.

When you see the 0 symbol for cardinality, you can assume two things:

i. The FK in the Order table allows nulls, and

ii. The FK is not part of the PK since PKs must not contain null values.

Figure 7.16: The relationship between a Customer table and an Order table, by A. Watt.
Video lecture:

https://tinyurl.com/j7a3jo5

Activity

Activity 7.0
Integrity Rules and Constraints

**Motivation:** Demonstrate the basic concepts on data integrity and constraints.

**Resources:** Internet access and Unit 7 learning materials.

**What to do:**
Read the following description and then answer questions 1-5 at the end.

The swim club database in Figure 7.17 has been designed to hold information about students who are enrolled in swim classes. The following information is stored: students, enrollment, swim classes, pools where classes are held, instructors for the classes, and various levels of swim classes. Use Figure 7.17 to answer questions 1 to 5.
Figure 7.17. ERD for questions 1-5. (Diagram by A. Watt.)

The primary keys are identified below. The following data types are defined in the SQL Server.

**tblLevels**
Level – Identity PK
ClassName – text 20 – nulls are not allowed

**tblPool**
Pool – Identity PK
PoolName – text 20 – nulls are not allowed
Location – text 30

**tblStaff**
StaffID – Identity PK
FirstName – text 20
MiddleInitial – text 3
LastName – text 30
Suffix – text 3
Salaried – Bit
PayAmount – money

**tblClasses**
LessonIndex – Identity PK
Level – Integer FK
SectionID – Integer
<table>
<thead>
<tr>
<th>Semester – TinyInt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days – text 20</td>
</tr>
<tr>
<td>Time – datetime (formatted for time)</td>
</tr>
<tr>
<td>Pool – Integer FK</td>
</tr>
<tr>
<td>Instructor – Integer FK</td>
</tr>
<tr>
<td>Limit – TinyInt</td>
</tr>
<tr>
<td>Enrolled – TinyInt</td>
</tr>
<tr>
<td>Price – money</td>
</tr>
</tbody>
</table>

**tblEnrollment**

<table>
<thead>
<tr>
<th>LessonIndex – Integer FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID – Integer FK (LessonIndex and SID) Primary Key</td>
</tr>
<tr>
<td>Status – text 30</td>
</tr>
<tr>
<td>Charged – bit</td>
</tr>
<tr>
<td>AmountPaid – money</td>
</tr>
<tr>
<td>DateEnrolled – datetime</td>
</tr>
</tbody>
</table>

**tblStudents**

<table>
<thead>
<tr>
<th>SID – Identity PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstName – text 20</td>
</tr>
<tr>
<td>MiddleInitial – text 3</td>
</tr>
<tr>
<td>LastName – text 30</td>
</tr>
<tr>
<td>Suffix – text 3</td>
</tr>
<tr>
<td>Birthday – datetime</td>
</tr>
<tr>
<td>LocalStreet – text 30</td>
</tr>
<tr>
<td>LocalCity – text 20</td>
</tr>
<tr>
<td>LocalPostalCode – text 6</td>
</tr>
<tr>
<td>LocalPhone – text 10</td>
</tr>
</tbody>
</table>

Implement this schema in SQL Server or access (you will need to pick comparable data types). Submit a screenshot of your ERD in the database.

1. Explain the relationship rules for each relationship (e.g., tblEnrollment and tblStudents: A student can enroll in many classes).

2. Identify cardinality for each relationship, assuming the following rules:
   i. A pool may or may not ever have a class.
   ii. The levels table must always be associated with at least one class.
   iii. The staff table may not have ever taught a class.
   iv. All students must be enrolled in at least one class.
| v. | The class must have students enrolled in it. |
| vi. | The class must have a valid pool. |
| vii. | The class may not have an instructor assigned. |

The class must always be associated with an existing level.

i) Which tables are weak and which tables are strong (covered in an earlier chapter)?

ii) Which of the tables are non-identifying and which are identifying

**Duration:** Expect to spend about 2 hour on this activity.

**Feedback:** This activity should be submitted to the course instructor for assessment and feedback.
Unit summary

In this unit you learned about Data Integrity and Constraints. We had a chance to learn about entity integrity and referential integrity. In addition, the unit content has also covered different types of relationships. Data Integrity and Constraints concepts assist the database designer especially in relational databases to ensure that the data is accurate and consistency.

References and Further Reading


Download this book for free at http://open.bccampus.ca

Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book *Database Design* by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Summary
Unit 8

Relational Design and Redundancy

Introduction

Generally, a good relational database design must capture all of the necessary attributes and associations. The design should do this with a minimal amount of stored information and no redundant data. In database design, redundancy is generally undesirable because it causes problems maintaining consistency after updates. In this unit, we will be introduced to the basic concepts of data redundancy in relation to database design and data anomalies.

Unit Outcomes

Upon completion of this unit you should be able to:

- Explain the basic concepts of data redundancy in database design.
- Compare and contrast different types of data anomalies.
- Apply various techniques in removing data anomalies during database design.

Terminologies

Deletion anomaly: Occurs when you delete a record that may contain attributes that shouldn’t be deleted.

Functional Dependency (FD): A describes how individual attributes are related

Insertion anomaly: Occurs when you are inserting inconsistent information into a table.

Join: Used when you need to obtain information based on two related tables.
8.1 Data Redundancy

Generally, a good relational database design must capture all of the necessary attributes and associations. The design should do this with a minimal amount of stored information and no redundant data.

In database design, redundancy is generally undesirable because it causes problems maintaining consistency after updates. However, redundancy can sometimes lead to performance improvements; for example, when redundancy can be used in place of a join to connect data. A join is used when you need to obtain information based on two related tables.

Consider table 8.1 (Example of ‘Customer’ bank account details in a certain bank) where ‘customer 1313131’ is displayed twice, once for account no. A-101 and again for account A-102. In this case, the customer number is not redundant, although there are deletion anomalies with the table. Having a separate customer table would solve this problem. However, if a branch address were to change, it would have to be updated in multiple places. If the customer number was left in the table as is, then you wouldn’t need a branch table and no join would be required, and performance is improved. Table 8.1 below illustrates redundancy used with bank accounts and branches.

Table 8.1: Bank Accounts

<table>
<thead>
<tr>
<th>AccountNo</th>
<th>Bal</th>
<th>Customer</th>
<th>Branch</th>
<th>Address</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-305</td>
<td>350</td>
<td>1234567</td>
<td>RoundHill</td>
<td>Horseneck</td>
<td>8000000</td>
</tr>
<tr>
<td>A-101</td>
<td>500</td>
<td>1313131</td>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
</tr>
<tr>
<td>A-102</td>
<td>400</td>
<td>1313131</td>
<td>Perryridge</td>
<td>Horseneck</td>
<td>1700000</td>
</tr>
<tr>
<td>A-113</td>
<td>600</td>
<td>9876543</td>
<td>Roundhill</td>
<td>Horseneck</td>
<td>8000000</td>
</tr>
<tr>
<td>A-201</td>
<td>900</td>
<td>9876543</td>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7100000</td>
</tr>
<tr>
<td>A-215</td>
<td>700</td>
<td>1111111</td>
<td>Manus</td>
<td>Horseneck</td>
<td>400000</td>
</tr>
<tr>
<td>A-222</td>
<td>700</td>
<td>1111111</td>
<td>Redwood</td>
<td>Palo Ato</td>
<td>2100000</td>
</tr>
</tbody>
</table>
8.2 Data Anomalies

Data anomalies are problems that can occur in poorly planned databases where all data and/or information are stored in one file. An example is a file-based systems. There are three types of data anomalies; namely, insertion anomaly, deletion anomaly and update anomaly.

8.2.1 Insertion Anomaly

An insertion anomaly occurs when you are inserting inconsistent information into a table. When we insert a new record, such as account no. A-306 in table 8.2, we need to check that the branch data is consistent with existing rows.

<table>
<thead>
<tr>
<th>AccountNo</th>
<th>Bal</th>
<th>Customer</th>
<th>Branch</th>
<th>Address</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>500</td>
<td>1313131</td>
<td>Downtown Brooklyn</td>
<td>9000000</td>
<td></td>
</tr>
<tr>
<td>A-102</td>
<td>400</td>
<td>1313131</td>
<td>Perryridge Horseneck</td>
<td>1700000</td>
<td></td>
</tr>
<tr>
<td>A-113</td>
<td>600</td>
<td>9876543</td>
<td>Roundhill Horseneck</td>
<td>8000000</td>
<td></td>
</tr>
<tr>
<td>A-201</td>
<td>900</td>
<td>9876543</td>
<td>Brighton Brooklyn</td>
<td>7100000</td>
<td></td>
</tr>
<tr>
<td>A-215</td>
<td>700</td>
<td>1111111</td>
<td>Manus Horseneck</td>
<td>400000</td>
<td></td>
</tr>
<tr>
<td>A-222</td>
<td>700</td>
<td>1111111</td>
<td>Redwood Palo Ato</td>
<td>2100000</td>
<td></td>
</tr>
<tr>
<td>A-305</td>
<td>350</td>
<td>1234567</td>
<td>RoundHill Horseneck</td>
<td>8000000</td>
<td></td>
</tr>
<tr>
<td>A-306</td>
<td>800</td>
<td>1111111</td>
<td>Roundhill Horseneck</td>
<td>8000000</td>
<td></td>
</tr>
</tbody>
</table>

8.2.2 Update Anomaly

If a branch changes address, such as the Round Hill branch in table 8.3, we need to update all rows referring to that branch. Changing existing information incorrectly is called an update anomaly.

<table>
<thead>
<tr>
<th>AccountNo</th>
<th>Bal</th>
<th>Customer</th>
<th>Branch</th>
<th>Address</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>500</td>
<td>1313131</td>
<td>Downtown Brooklyn</td>
<td>9000000</td>
<td></td>
</tr>
<tr>
<td>A-102</td>
<td>400</td>
<td>1313131</td>
<td>Perryridge Horseneck</td>
<td>1700000</td>
<td></td>
</tr>
</tbody>
</table>
8.2.3 Deletion Anomaly

A deletion anomaly occurs when you delete a record that may contain attributes that shouldn’t be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in table 8.4, all of the branch information disappears. Compare and contrast the two tables below which illustrate the concept of deletion anomaly. Table 8.4 (before deletion of account A-101) and table 8.5 (after deletion of account A-101).

Table 8.4: Before deletion of A-101

<table>
<thead>
<tr>
<th>AccountNo</th>
<th>Bal</th>
<th>Customer</th>
<th>Branch</th>
<th>Address</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>500</td>
<td>1313131</td>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
</tr>
<tr>
<td>A-102</td>
<td>400</td>
<td>1313131</td>
<td>Perryridge</td>
<td>Horseneck</td>
<td>1700000</td>
</tr>
<tr>
<td>A-113</td>
<td>600</td>
<td>9876543</td>
<td>Roundhill</td>
<td>Horseneck</td>
<td>8000000</td>
</tr>
<tr>
<td>A-201</td>
<td>900</td>
<td>9876543</td>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7100000</td>
</tr>
<tr>
<td>A-215</td>
<td>700</td>
<td>1111111</td>
<td>Manus</td>
<td>Horseneck</td>
<td>400000</td>
</tr>
<tr>
<td>A-222</td>
<td>700</td>
<td>1111111</td>
<td>Redwood</td>
<td>Palo Ato</td>
<td>2100000</td>
</tr>
<tr>
<td>A-305</td>
<td>350</td>
<td>1234567</td>
<td>RoundHill</td>
<td>Horseneck</td>
<td>8000000</td>
</tr>
</tbody>
</table>
Table 8.5: Illustration of deletion anomaly

<table>
<thead>
<tr>
<th>AccountNo</th>
<th>Bal</th>
<th>Customer</th>
<th>Branch</th>
<th>Address</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-102</td>
<td>400</td>
<td>1313131</td>
<td>Perryridge</td>
<td>Horseneck</td>
<td>1700000</td>
</tr>
<tr>
<td>A-113</td>
<td>600</td>
<td>9876543</td>
<td>Roundhill</td>
<td>Horseneck</td>
<td>8000000</td>
</tr>
<tr>
<td>A-201</td>
<td>900</td>
<td>9876543</td>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7100000</td>
</tr>
<tr>
<td>A-215</td>
<td>700</td>
<td>1111111</td>
<td>Manus</td>
<td>Horseneck</td>
<td>400000</td>
</tr>
<tr>
<td>A-222</td>
<td>700</td>
<td>1111111</td>
<td>Redwood</td>
<td>Palo Ato</td>
<td>2100000</td>
</tr>
<tr>
<td>A-305</td>
<td>350</td>
<td>1234567</td>
<td>RoundHill</td>
<td>Horseneck</td>
<td>800000</td>
</tr>
</tbody>
</table>

The problem with deleting the A-101 row is we don’t know where the Downtown branch is located and we lose all information regarding customer 1313131. To avoid these kinds of update or deletion problems, we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables.

Each bank account table must contain information about one entity only, such as the Branch or Customer, as displayed in Figure 10.5.

Following this practice will ensure that when branch information is added or updated it will only affect one record. So, when customer information is added or deleted, the branch information will not be accidentally modified or incorrectly recorded.
Example: employee project table and anomalies

Table 8.6 shows an example of an employee project table. From this table, we can assume that:

i. EmpID and ProjectID are a composite PK.

ii. Project ID determines Budget (i.e., Project P1 has a budget of 32 hours).

Table 8.6: Example of an employee-project table

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Budget</th>
<th>ProjectID</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>S75</td>
<td>32</td>
<td>P1</td>
<td>7</td>
</tr>
<tr>
<td>S75</td>
<td>40</td>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>S79</td>
<td>32</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>S79</td>
<td>27</td>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>S80</td>
<td>40</td>
<td>P2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>P4</td>
<td></td>
</tr>
</tbody>
</table>

Next, let’s look at some possible anomalies that might occur with this table during the following steps:

i. Action: Add row \{S85,35,P1,9\}

ii. Problem: There are two tuples with conflicting budgets

iii. Action: Delete tuple \{S79, 27, P3, 1\}

iv. Problem: Step #3 deletes the budget for project P3

v. Action: Update tuple \{S75, 32, P1, 7\} to \{S75, 35, P1, 7\}

vi. Problem: Step #5 creates two tuples with different values for project P1’s budget

vii. Solution: Create a separate table, each, for Projects and Employees, as shown in Figure 8.2
8.3 How to Avoid Anomalies

The best approach to creating tables without anomalies is to ensure that the tables are normalized, and that’s accomplished by understanding functional dependencies. FD ensures that all attributes in a table belong to that table. In other words, it will eliminate redundancies and anomalies. Example: separate Project and Employee tables.

Table 8.7: Project details table

<table>
<thead>
<tr>
<th>ProjectID</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>32</td>
</tr>
<tr>
<td>P2</td>
<td>40</td>
</tr>
<tr>
<td>P3</td>
<td>27</td>
</tr>
<tr>
<td>P4</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 8.8: Employee details table

<table>
<thead>
<tr>
<th>EmpID</th>
<th>ProjectID</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>S75</td>
<td>P1</td>
<td>7</td>
</tr>
<tr>
<td>S75</td>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>S79</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>S79</td>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>S80</td>
<td>P2</td>
<td>5</td>
</tr>
</tbody>
</table>
Tables 8.7 and 8.8 respectively, separate Project and Employee tables with data. By keeping data separate using individual Project and Employee tables:

i. No anomalies will be created if a budget is changed.

ii. No dummy values are needed for projects that have no employees assigned.

iii. If an employee’s contribution is deleted, no important data is lost.

iv. No anomalies are created if an employee’s contribution is added.

---

**Video lecture:**

[https://tinyurl.com/zpgydth](https://tinyurl.com/zpgydth)
Activity

Activity 8.0
Data Anomalies

Aim: Demonstrate basic techniques of how to remove data anomalies.

Resources: Unit 8 learning materials.

What to do:
By using the given data below, please remove all the data anomalies involved.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Sample Value</th>
<th>Sample Value</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudentID</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>StudentName</td>
<td>John Smith</td>
<td>Sandy Law</td>
<td>Sue Rogers</td>
</tr>
<tr>
<td>CourseID</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>CourseName</td>
<td>Programming Level 1</td>
<td>Programming Level 1</td>
<td>Business</td>
</tr>
<tr>
<td>Grade</td>
<td>75%</td>
<td>61%</td>
<td>81%</td>
</tr>
<tr>
<td>CourseDate</td>
<td>Jan 5th, 2014</td>
<td>Jan 5th, 2014</td>
<td>Jan 7th, 2014</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 1 hour on this activity.

Feedback: This activity should be submitted to the course instructor for assessment and feedback.

Unit summary

In this unit you learned about data redundancy in relation with database design; specifically covering the three major types of data anomalies. The unit concluded by illustrating how to remove them.
Review Questions

Aim: These review questions are aimed at assisting the students in understanding the concept of data redundancy and data anomalies. Therefore, it is advised that they should be posted in a discussion forum where each student can contribute.

1. What is meaning of the term ‘data redundancy’? Explain in details why this concept is a problem in relational database?
2. Why is it important to remove data redundancy before continuing with database design and implementation?
3. Compare and contrast between update and insertion anomalies.
4. Explain in details the concept of deletion anomaly.
5. Describe the mechanism of removing data anomalies before database implementation.

References and Further Reading


Download this book for free at http://open.bccampus.ca
Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Unit Summary
3. Review questions
Unit 9

Functional Dependencies

Introduction

In this unit, we will be introduced to the concept of Functional Dependency (FD) which is an important part of relational database design and a pre-requisite for the next unit (normalization). FD in relational databases defines relationship between two sets of attributes. Therefore, we will learn more on functional dependencies categories.

Unit Outcomes

Upon completion of this unit you should be able to:

- Describe the concept of functional dependency.
- Compare and contrast different types of functional dependencies.
- Identify various types of inference rules in relation to FD.
- Use n dependency diagram tool to define FD.

Terminologies

Armstrong’s axioms: A set of inference rules used to infer all the functional dependencies on a relational database.

Decomposition: A rule that suggests if you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables.

Dependent: The right side of the functional dependency diagram.

Determinant: It is the left side of the functional dependency diagram.

Functional Dependency (FD): A relationship between two attributes, typically between the PK and other non-key attributes within a table.
9.1 Functional Dependency

A functional dependency (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below:

\[ X \rightarrow Y \]

The left side of the above FD diagram is called the determinant, and the right side is the dependent. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

\[ \text{SIN} \rightarrow \text{Name, Address, Birthdate} \]

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

\[ \text{SIN, Course} \rightarrow \text{DateCompleted} \]

The third example indicates that ISBN determines Title.

\[ \text{ISBN} \rightarrow \text{Title} \]

9.2 Rules of Functional Dependencies

Consider the following table of data \( r(R) \) of the relation schema R(ABCDE) shown in Table 9.1 below:

Table 9.1: Functional dependency

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>l</td>
<td>b</td>
<td>c</td>
<td>l</td>
</tr>
</tbody>
</table>
As you look at this table, ask yourself: What kind of dependencies can we observe among the attributes in Table R? Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

A → B,  A → C,  A → D,  A → E

1. It also follows that A → BC (or any other subset of ABCDE).
2. This can be summarized as A → BCDE.
3. From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

A → E,  B → E,  C → E,  D → E

However, we cannot generally summarize the above with ABCD → E because, in general, A → E, B → E, AB → E.

Other observations:

1. Combinations of BC are unique, therefore BC → ADE.
2. Combinations of BD are unique, therefore BD → ACE.
3. If C values match, so do D values.
   i. Therefore, C → D
   ii. However, D values don’t determine C values
   iii. So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

9.3.1 Inference Rules

Armstrong’s axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.
Let $R(U)$ be a relation scheme over the set of attributes $U$. We will use the letters $X$, $Y$, $Z$ to represent any subset of and, for short, the union of two sets of attributes, instead of the usual $X \cup Y$.

i) **Axiom of reflexivity**

This axiom says, if $Y$ is a subset of $X$, then $X$ determines $Y$ (see Figure 9.1).

$$\text{If } Y \subseteq X \text{, then } X \rightarrow Y$$

Figure 9.1: Equation for axiom of reflexivity, by A. Watt.

For example, $\text{PartNo} \rightarrow \text{NT123}$ where $X$ (PartNo) is composed of more than one piece of information; i.e., $Y$ (NT) and partID (123).

ii) **Axiom of augmentation**

The axiom of augmentation, also known as a partial dependency, says if $X$ determines $Y$, then $XZ$ determines $YZ$ for any $Z$ (see Figure 11.2).

$$\text{If } X \rightarrow Y \text{, then } XZ \rightarrow YZ \text{ for any } Z$$

Figure 9.2: Equation for axiom of augmentation, by A. Watt

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK. In the example shown below, StudentName, Address, City, Prov, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Grade.

$\text{StudentNo, Course} \rightarrow \text{StudentName, Address, City, Prov, PC, Grade, DateCompleted}$

This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).
To fix this problem, we need to break the original table down into two as follows:

- Table 1: StudentNo, Course, Grade, DateCompleted
- Table 2: StudentNo, StudentName, Address, City, Prov, PC

iii) Axiom of transitivity

The axiom of transitivity says if $X$ determines $Y$, and $Y$ determines $Z$, then $X$ must also determine $Z$ (see Figure 11.3).

![Figure 9.3: Equation for axiom of transitivity, by A. Watt](image)

The table below has information not directly related to the student; for instance, ProgramID and ProgramName should have a table of its own. ProgramName is not dependent on StudentNo; it’s dependent on ProgramID.

```
StudentNo —> StudentName, Address, City, Prov, PC, ProgramID, ProgramName
```

This situation is not desirable because a non-key attribute (ProgramName) depends on another non-key attribute (ProgramID).

To fix this problem, we need to break this table into two: one to hold information about the student and the other to hold information about the program.

- Table 1: StudentNo —> StudentName, Address, City, Prov, PC, ProgramID
- Table 2: ProgramID —> ProgramName

However we still need to leave an FK in the student table so that we can identify which program the student is enrolled in.
iv) Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together.

It states that if $X$ determines $Y$ and $X$ determines $Z$ then $X$ must also determine $Y$ and $Z$ (see Figure 11.4).

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

Figure 9.4: Equation for the Union rule, by A. Watt

For example, if:

- $SIN \rightarrow \text{EmpName}$
- $SIN \rightarrow \text{SpouseName}$

You may want to join these two tables into one as follows:

$SIN \rightarrow \text{EmpName, SpouseName}$

Some database administrators (DBA) might choose to keep these tables separated for a couple of reasons. One, each table describes a different entity so the entities should be kept apart. Two, if SpouseName is to be left NULL most of the time, there is no need to include it in the same table as EmpName.

v) Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if $X$ determines $Y$ and $Z$, then $X$ determines $Y$ and $X$ determines $Z$ separately (see Figure 11.5).

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

Figure 9.5: Equation for decomposition rule, by A. Watt
9.2 Dependency Diagram

A dependency diagram, shown in Figure 9.6, illustrates the various dependencies that might exist in a non-normalized table. A non-normalized table is one that has data redundancy in it.

![Dependency Diagram](image)

**Figure 9.6: Dependency diagram, by A. Watt**

The following dependencies are identified in this table:

i. ProjectNo and EmpNo, combined, are the PK.

ii. Partial Dependencies:

- ProjectNo —>ProjName
- EmpNo —>EmpName, DeptNo,
- ProjectNo, EmpNo —>HrsWork
- Transitive Dependency:
  - DeptNo —>DeptName
Activity

**Activity 9.0**

Functional Dependency Exercises

**Aim:** Demonstrate basic knowledge of Functional Dependencies Concepts.

**Resources:** Internet access and Unit 8 learning materials.

**What to do:**

1. By using the definition of functional dependency to confirm that each of Armstrong’s axioms (reflexivity, augmentation and transitivity) is correct.

2. In the following table, identify two functional dependencies (A, B and C are attributes)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
</tr>
</tbody>
</table>

3. Express the following real world facts using functional dependencies: A lecturer, identified by the value of the attribute LecturerId, has a name (Name), an office (Office), and a phone extension number (ExtensionNo).

   The number of students (NoOfStud) enrolled in a course, which is identified by the value of the attribute CourseId, depends on the term when the course is offered.
(Term) and the year (Year).

Each office (Office) has only one phone extension number (ExtensionNo) and each phone extension number belongs to at most one office.

**Duration:** Expect to spend about 1 hour on this activity.

**Feedback:** This is an activity should be submitted to the course instructor for assessment and feedback.

---

**Unit summary**

In this unit you learned about Functional Dependencies (FD). The unit has also covered different inference rules and dependency diagram. The concepts covered in this unit are a pre-requisite for the next unit of data normalization.

**Review Questions**

**Motivation:** These review questions are aimed at assisting the students in understanding of functional dependencies concepts. Therefore, it is advised that they should be posted in a discussion forum where each student can contribute.

**Part I**

Consider a relation $R$ with five attributes $ABCDE$. You are given the following dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$.

i. List all keys for $R$.

ii. Is $R$ in 3NF?

iii. Is $R$ in BCNF?
Part II

i. Define the term *functional dependency* and give an example to illustrate.

ii. Why are some functional dependencies called *trivial*?

Part III

Bank G wants to have a database for a bank that contains accounts (C), branches (H) and customers (D). Detailed below are the requirements given the following constraints:

i. An account cannot be shared by multiple customers.

ii. Two different branches do not have the same account.

iii. Each customer can have at most one account in a branch (but different accounts in different branches).

Write the functional dependencies implied by the above mentioned requirements.

Part IV

Identify the functional dependencies in the following relations and represent the dependencies using the proper notation.

i. Course (CourseCode, CourseName, CourseUnits)

ii. Instructor(InstrnID, InstrName, InstrAge, CourseCode)
References and Further Reading


Download this book for free at http://open.bccampus.ca

Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Unit Summary
3. Review questions
Unit 10

Introduction to Data Normalization

Introduction

In the previous units, we have covered theories on data modeling, relational data modeling and entity relationship model which logical design for relational databases. In this unit, we will learn that for any data in database table must be stored in a normalized way. This unit contents will cover the properties of a normalized table, process of normalization and its importance to the structure of a database.

Unit Outcomes

Upon completion of this unit you will be able to:

- Describe the fundamental concepts of normalization.
- Compare and Contrast different types of normal forms.
- Normalize a database relation to a third normal form.

Terminologies

Normalization: The process of determining how much redundancy exists in a database table.

1NF: First Normal Forms - only single values are permitted at the intersection of each row and column so there are no repeating groups.

2NF: Second Normal Form - the relation must be in 1NF and the PK comprises a single attribute.

3NF: Third Normal Form - the relation must be in 2NF and all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute.

Normalization: The process of determining how much redundancy exists in a database table.
10.1 Overview

Normalization is the branch of relational theory that provides design insights. It is the process of determining how much redundancy exists in a table. The goals of normalization are to:

- Be able to characterize the level of redundancy in a relational schema.
- Provide mechanisms for transforming schemas in order to remove redundancy.

Normalization theory draws heavily on the theory of functional dependencies. Normalization theory defines six normal forms (NF). Each normal form involves a set of dependency properties that a schema must satisfy and each normal form gives guarantees about the presence and/or absence of update anomalies. This means that higher normal forms have less redundancy, and as a result, fewer update problems.

Normalization should be part of the database design process. However, it is difficult to separate the normalization process from the ER modeling process so the two techniques should be used concurrently as follows:

- Use an entity relation diagram (ERD) to provide the big picture, or macro view, of an organization’s data requirements and operations. This is created through an iterative process that involves identifying relevant entities, their attributes and their relationships.

- Normalization procedure focuses on characteristics of specific entities and represents the micro view of entities within the ERD.

10.2 Normal Forms

All the tables in any database can be in one of the normal forms we will discuss next. Ideally we only want minimal redundancy for PK to FK. Everything else should be derived from other tables. There are six normal forms, but we will only look at the first four, which are:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
• Boyce-Codd normal form (BCNF)

BCNF is rarely used.

10.3.1 First Normal Form (1NF)

In the *first normal form*, only single values are permitted at the intersection of each row and column; hence, there are no repeating groups.

To normalize a relation that contains a repeating group, remove the repeating group and form two new relations. The PK of the new relation is a combination of the PK of the original relation plus an attribute from the newly created relation for unique identification.

**Process for 1NF**

We will use the Student_Grade_Report table below, from a School database, as our example to explain the process for 1NF.

**Student_Grade_Report (StudentNo, StudentName, Major, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)**

- In the Student Grade Report table, the repeating group is the course information. A student can take many courses.
- Remove the repeating group. In this case, it’s the course information for each student.
- Identify the PK for your new table.
- The PK must uniquely identify the attribute value (StudentNo and CourseNo).
- After removing all the attributes related to the course and student, you are left with the student course table (StudentCourse).
- The Student table (Student) is now in first normal form with the repeating group removed.

The two new tables are shown below.

- **Student (StudentNo, StudentName, Major)**
- **StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)**
i) How to update 1NF anomalies

StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- To add a new course, we need a student.
- When course information needs to be updated, we may have inconsistencies.
- To delete a student, we might also delete critical information about a course.

10.3.2 Second normal form (2NF)

For the second normal form, the relation must first be in 1NF. The relation is automatically in 2NF if, and only if, the PK comprises a single attribute.

If the relation has a composite PK, then each non-key attribute must be fully dependent on the entire PK and not on a subset of the PK (i.e., there must be no partial dependency or augmentation).

ii) Process for 2NF

To move to 2NF, a table must first be in 1NF.

- The Student table is already in 2NF because it has a single-column PK.
- When examining the Student Course table, we see that not all the attributes are fully dependent on the PK; specifically, all course information. The only attribute that is fully dependent is grade.
- Identify the new table that contains the course information.
- Identify the PK for the new table.

The three new tables are shown below.

- **Student** (StudentNo, StudentName, Major)
- **CourseGrade** (StudentNo, CourseNo, Grade)
- **CourseInstructor** (CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation)
iii) How to update 2NF anomalies

- When adding a new instructor, we need a course.
- Updating course information could lead to inconsistencies for instructor information.
- Deleting a course may also delete instructor information.

10.3.3 Third normal form (3NF)

To be in third normal form, the relation must be in second normal form. Also all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute.

i) Process for 3NF

The following are the stages involved in normalizing data to 3NF:-

- Eliminate all dependent attributes in transitive relationship(s) from each of the tables that have a transitive relationship.
- Create new table(s) with removed dependency.
- Check new table(s) as well as table(s) modified to make sure that each table has a determinant and that no table contains inappropriate dependencies.

See the four new tables below.

- **Student** (StudentNo, StudentName, Major)
- **CourseGrade** (StudentNo, CourseNo, Grade)
- **Course** (CourseNo, CourseName, InstructorNo)
- **Instructor** (InstructorNo, InstructorName, InstructorLocation)

At this stage, there should be no anomalies in third normal form. Let’s look at the dependency diagram (Figure 10.1) for this example. The first step is to remove repeating groups, as discussed above.

- **Student** (StudentNo, StudentName, Major)
- **StudentCourse** (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

To recap the normalization process for the School database, review the dependencies shown in Figure 10.1.
The abbreviations used in Figure 10.1 are as follows:

i. PD: partial dependency

ii. TD: transitive dependency

iii. FD: full dependency (Note: FD typically stands for functional dependency. Using FD as an abbreviation for full dependency is only used in Figure 10.1.)

10.3.4 Boyce-Codd Normal Form (BCNF)

When a table has more than one candidate key, anomalies may result even though the relation is in 3NF. Boyce-Codd normal form is a special case of 3NF. A relation is in BCNF if, and only if, every determinant is a candidate key.

**BCNF Example 1**- Consider the following table 10.1 below;

<table>
<thead>
<tr>
<th>Student_id</th>
<th>Major</th>
<th>Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Physics</td>
<td>Smith</td>
</tr>
<tr>
<td>111</td>
<td>Music</td>
<td>Chan</td>
</tr>
<tr>
<td>320</td>
<td>Math</td>
<td>Dobbs</td>
</tr>
<tr>
<td>671</td>
<td>Physics</td>
<td>White</td>
</tr>
<tr>
<td>803</td>
<td>Physics</td>
<td>Smith</td>
</tr>
</tbody>
</table>
The semantic rules (business rules applied to the database) for this table are:

i. Each Student may major in several subjects.

ii. For each Major, a given Student has only one Advisor.

iii. Each Major has several Advisors.

iv. Each Advisor advises only one Major.

v. Each Advisor advises several Students in one Major.

The functional dependencies for this table are listed below. The first one is a candidate key; the second is not.

i. Student_id, Major → Advisor

ii. Advisor → Major

Anomalies for this table include:

i. Delete – student deletes advisor info

ii. Insert – a new advisor needs a student

iii. Update – inconsistencies

Note: No single attribute is a candidate key.

PK can be Student_id, Major or Student_id, Advisor.

To reduce the St_Maj_Adv relation to BCNF, you create two new tables as illustrated below:

i. St_Adv (Student_id, Advisor)

<table>
<thead>
<tr>
<th>Student_id</th>
<th>Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Smith</td>
</tr>
<tr>
<td>111</td>
<td>Chan</td>
</tr>
<tr>
<td>320</td>
<td>Dobbs</td>
</tr>
<tr>
<td>671</td>
<td>White</td>
</tr>
<tr>
<td>803</td>
<td>Smith</td>
</tr>
</tbody>
</table>
ii. **Adv Maj** (Advisor, Major)

<table>
<thead>
<tr>
<th>Advisor</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Physics</td>
</tr>
<tr>
<td>Chan</td>
<td>Music</td>
</tr>
<tr>
<td>Dobbs</td>
<td>Math</td>
</tr>
<tr>
<td>White</td>
<td>Physics</td>
</tr>
</tbody>
</table>

**BCNF Example 2**

Consider the following table 10.4 below;

<table>
<thead>
<tr>
<th>ClientNo</th>
<th>InterviewDate</th>
<th>InterviewTime</th>
<th>StaffNo</th>
<th>RoomNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR76</td>
<td>13-May-02</td>
<td>10.30</td>
<td>SG5</td>
<td>G101</td>
</tr>
<tr>
<td>CR56</td>
<td>13-May-02</td>
<td>12.00</td>
<td>SG5</td>
<td>G101</td>
</tr>
<tr>
<td>CR74</td>
<td>13-May-02</td>
<td>12.00</td>
<td>SG37</td>
<td>G102</td>
</tr>
<tr>
<td>CR56</td>
<td>1-July-02</td>
<td>10.30</td>
<td>SG5</td>
<td>G102</td>
</tr>
</tbody>
</table>

i. FD1 – ClientNo, InterviewDate –> InterviewTime, StaffNo, RoomNo (PK)

ii. FD2 – staffNo, interviewDate, interviewTime –> clientNO (candidate key: CK)

iii. FD3 – roomNo, interviewDate, interviewTime –> staffNo, clientNo (CK)

iv. FD4 – staffNo, interviewDate –> roomNo

A relation is in BCNF if, and only if, every determinant is a candidate key. We need to create a table that incorporates the first three FDs table 10.5 (**Client Interview2**) and table 10.6 (**StaffRoom**) for the fourth FD.

<table>
<thead>
<tr>
<th>ClientNo</th>
<th>InterviewDate</th>
<th>InterviewTime</th>
<th>StaffNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR76</td>
<td>13-May-02</td>
<td>10.30</td>
<td>SG5</td>
</tr>
<tr>
<td>CR56</td>
<td>13-May-02</td>
<td>12.00</td>
<td>SG5</td>
</tr>
<tr>
<td>CR74</td>
<td>13-May-02</td>
<td>12.00</td>
<td>SG37</td>
</tr>
<tr>
<td>CR56</td>
<td>1-July-02</td>
<td>10.30</td>
<td>SG5</td>
</tr>
</tbody>
</table>
10.3 Normalization and Database Design

During the normalization process of database design, make sure that proposed entities meet required normal form before table structures are created. Many real-world databases have been improperly designed or burdened with anomalies if improperly modified during the course of time. You may be asked to redesign and modify existing databases. This can be a large undertaking if the tables are not properly normalized.

Video lecture:

https://tinyurl.com/jc6dbyp

https://tinyurl.com/gu49pgj
Activity

**Activity 10.1**
Introduction to Normalization Scenarios

**Motivation:** To become conversant with basic techniques of normalizing data.

**Resources:** Unit 10 learning materials.

**What to do:**
Read the following scenarios carefully;

1. **BRANCH** *(Branch#, Branch_Addr, (ISBN, Title, Author, Publisher, Num_copies))*

2. **CLIENT** *(Client#, Name, Location, Manager#, Manager_name, Manager_location, (Contract#, Estimated_cost, Completion_date, (Staff#, Staff_name, Staff_location)))*

3. **PATIENT** *(Patient#, Name, DOB, Address, (Prescription#, Drug, Date, Dosage, Doctor#, Doctor, Secretary))*

4. **DOCTOR** *(Doctor#, DoctorName, Secretary, (Patient#, PatientName, PatientDOB, PatientAddress, (Prescription#, Drug, Date, Dosage)))*

**How to do it:**
You are provided with different scenarios; please normalize the data to 3NF.

**Duration:** Expect to spend about 2 hours on this activity

**Feedback:** This activity should be submitted to the course Instructor for assessment and feedback.
Unit 10 Introduction to Data Normalization

Unit summary

In this unit you learned data normalization process which is used parallel with ER modelling in the logical design of database implementation. This unit also covered insert, update and delete anomalies which can be removed by applying normalization to that particular database. The last part covers the first three stages of data normalization, namely, 1NF, 2NF and 3NF.

Case Study

Motivation: The aim of these case studies is to assist you in understanding different case studies in database design and use the correct techniques in normalizing the data. The following are the case studies where students will read and then normalize the data given to 3NF.

Case Study I

A new client comes and asks for you to build them a database from the spreadsheet they've been using to track the company employees. Here's an excerpt:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Sally</td>
<td>Sales</td>
<td>Manager</td>
<td>Sally</td>
<td>$25</td>
<td>35</td>
<td>$875</td>
<td>N/A</td>
</tr>
<tr>
<td>Sally</td>
<td>Sales</td>
<td>Manager</td>
<td>Sally</td>
<td>$25</td>
<td>35</td>
<td>$875</td>
<td>N/A</td>
</tr>
<tr>
<td>Joe</td>
<td>Sales</td>
<td>Sales person</td>
<td>Sally</td>
<td>$10</td>
<td>35</td>
<td>$350</td>
<td>6%</td>
</tr>
<tr>
<td>Sean</td>
<td>Shipping</td>
<td>Clerk</td>
<td>Bob</td>
<td>$8</td>
<td>20</td>
<td>$160</td>
<td>N/A</td>
</tr>
<tr>
<td>Sean</td>
<td>Security</td>
<td>Guard</td>
<td>Kirk</td>
<td>$12</td>
<td>16</td>
<td>$192</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Case Study II

The ABC Manufacturing company has a completely automated application system. The system, however, resides on index files and does not allow for decision support at all. In order to move to ad hoc queries, and "what if" queries, the company has decided to convert the existing system to a database. Initially, the only criterion for the application is to replace the existing system with a database system. No ad hoc screen or reports have been anticipated. You will see the reports and screens that exist currently.

Customer Order and Product Application Considerations

1. Each customer must be on file before an order can be placed. The name, address(s), phone number(s), and credit limit must be recorded. All other data items are optional. If there is no shipping address, then the mailing address is used instead. Since customers can have identical names, a customer id has been assigned to each customer.

2. Each order will have a computer generated id number. The order can have up to 10 line items. Discounts can be given to preferred customers and this discount amount will be recorded on the customer's record. Customers without a discount amount will not be given a discount.

3. Each product listed on the order will show the standard price for that product. Discounts will be shown at the bottom of the order form.

4. Orders that can be filled or partially filled are shipped immediately, and the product data is updated accordingly. Orders, or partial orders that cannot be filled will be backordered.

5. As products are manufactured the product data is updated accordingly along with the part inventory data.

6. A customer can place numerous orders. Products can be ordered by many different customers. The same part can be used in numerous products. (eg. a screw can be used in a chair, bar stool etc.)
Case Study III

A college keeps details of its students, staff and courses in a file. Part of this file is shown below.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0567</td>
<td>J Evans</td>
<td>11/4/89</td>
<td>M</td>
<td>COMP7</td>
<td>Computing A Level</td>
<td>186</td>
<td>H Smith</td>
</tr>
<tr>
<td>8453</td>
<td>R Begum</td>
<td>18/3/88</td>
<td>F</td>
<td>BIOL9</td>
<td>Biology A Level</td>
<td>78</td>
<td>D Jones</td>
</tr>
<tr>
<td>0567</td>
<td>J Evans</td>
<td>11/4/89</td>
<td>M</td>
<td>MATH5</td>
<td>Maths A Level</td>
<td>186</td>
<td>H Smith</td>
</tr>
</tbody>
</table>

Key for field name:
1 = StudentNo, 2 = StudentName, 3 = DateOfBirth, 4 = Gender, 5 = Course No, 6 = Course Name, 7 = LecturerNo, 8 = Lecturer Name

The data in this file is not normalised.

i. Using data from the above file to illustrate your answer, describe two different problems associated with data not being normalised.

ii. The above data can be re-organised into a normalised relational database with tables linked using primary and foreign keys.

Re-organise this data into a normalised relational database using two tables. You should clearly indicate the table names and any primary or foreign keys that you use.

Feedback: These case studies should be submitted to the course Instructor for assessment and feedback.
References and Further Reading


   Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book *Database Design* by Adrienne Watt and Nelson Eng licensed under [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

Download this book for free at [http://open.bccampus.ca](http://open.bccampus.ca)

The following material was written by Grace Mbwete:

1. Introduction
2. Unit Summary
Unit 11

Introduction to SQL

Introduction

Structured Query Language (SQL) is the main data definition language used for the creation and maintenance of databases. In this unit, we will learn the basic SQL syntax, including some data definition commands, how to define different types of constraints and different user defined data types in the application environment of databases.

Unit Outcomes

Upon completion of this unit you will be able to:

- \textit{Describe} different SQL - DDL commands.
- \textit{Identify} where and when to use the appropriate SQL DDL commands.
- \textit{Create} database by using SQL DDL commands.
- \textit{Create} set of database tables using DDL commands.
- \textit{Describe} the various types of integrity constraints and how they are used.

Terminologies

\begin{itemize}
    \item \textbf{SQL:} Database language designed for managing data held in a relational database management system.
    \item \textbf{DDL:} Data Definition Language; standard for commands that define the different structures in a database
    \item \textbf{DML:} Set of syntax elements used for selecting, inserting, deleting and updating data in a database
    \item \textbf{Constraints:} Rules enforced on data columns on table
\end{itemize}
11.1 Introduction to SQL

*Structured Query Language* (SQL) is a database language designed for managing data held in a relational database management system. SQL was initially developed by IBM in the early 1970s. The initial version, called *SEQUEL* (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM’s quasi-relational database management system, System R. Then in the late 1970s, Relational Software Inc., which is now Oracle Corporation, introduced the first commercially available implementation of SQL, Oracle V2 for VAX computers.

Many of the currently available relational DBMSs, such as Oracle Database, Microsoft SQL Server (shown in Figure 15.1), MySQL, IBM DB2, IBM Informix and Microsoft Access, use SQL.

![Microsoft SQL Server Management Studio](image)

Figure 11.1: Example of Microsoft SQL Server, by A. Watt.

11.2.2 Data Definition Language (DDL) Commands

In a DBMS, the SQL database language is used to:

- Create the database and table structures
- Perform basic data management chores (add, delete and modify)
• Perform complex queries to transform raw data into useful information

In this unit, we will focus on using SQL to create the database and table structures, mainly using SQL as a data definition language (DDL). In the following unit; unit 12 we will use SQL as a data manipulation language (DML) to insert, delete, select and update data within the database tables.

11.2.1 Create Database Command

The major SQL DDL statements are CREATE DATABASE and CREATE/DROP/ALTER TABLE. The SQL statement CREATE is used to create the database and table structures. The SQL syntax for creating a database is as follows:-

`CREATE DATABASE dbname;`

**Example: CREATE DATABASE university**

A new database named ‘university’ is created by the SQL statement;

`CREATE DATABASE university;`

In order for your database to be created within a RDBMS; you must adhere to the following conditions, which are:

• Making sure you have admin privilege before creating any database.

• The SQL command (CREATE DATABASE) is in capital letters and the command ends with a semicolon sign (;).

• Ensuring that the database name should be unique within the RDBMS.

After creating the database ‘university’; you can now check your created database in the list of databases by typing the following SQL statement;

`SHOW DATABASES;`

The result of this SQL statement should list down the databases that have been created with ‘university’ database which has just been create; as follows:-
11.2.2 Create Table Command

A new database named ‘university’ is created by the SQL statement CREATE DATABASE university. Once the database is created, the next step is to create the database tables. The general format for the CREATE TABLE command is:

```
CREATE TABLE <tablename>
(
    ColumnName, Datatype, Optional Column Constraint,
    ColumnName, Datatype, Optional Column Constraint,
    Optional table Constraints
);
```

Tablename is the name of the database table such as Employee. Each field in the CREATE TABLE has three parts (see above):

- ColumnName
- Data type
- Optional Column Constraint

i) ColumnName

The ColumnName must be unique within the table. Some examples of ColumnNames are FirstName and LastName.

ii) Data Type

The data type, as described below, must be a system data type or a user-defined data type. Many of the data types have a size such as CHAR(35) or Numeric(8,2).

1. **Bit** – Integer data with either a 1 or 0 value
2. **Int** – Integer (whole number) data from \(-2^{31}\) (-2,147,483,648) through \(2^{31} - 1\) (2,147,483,647)

3. **Smallint** – Integer data from \(2^{15}\) (-32,768) through \(2^{15} - 1\) (32,767)

4. **Tinyint** – Integer data from 0 through 255

5. **Decimal** – Fixed precision and scale numeric data from \(-10^{38} - 1\) through \(10^{38}\)

6. **Numeric** – A synonym for decimal

7. **Timestamp** – A database-wide unique number

8. **Uniqueidentifier** – A globally unique identifier (GUID)

9. **Money** – Monetary data values from \(-2^{63}\) (-922,337,203,685,477.5808) through \(2^{63} - 1\) (+922,337,203,685,477.5807), with accuracy to one-tenthousandth of a monetary unit

10. **Smallmoney** – Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to one-tenthousandth of a monetary unit

11. **Float** – Floating precision number data from \(-1.79E + 308\) through \(1.79E + 308\)

12. **Real** – Floating precision number data from \(-3.40E + 38\) through \(3.40E + 38\)

13. **Datetime** – Date and time data from January 1, 1753, to December 31, 9999, with an accuracy of one-three-hundredths of a second, or 3.33 milliseconds

14. **Smalldatetime** – Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute

15. **Char** – Fixed-length non-Unicode character data with a maximum length of 8,000 characters

16. **Varchar** – Variable-length non-Unicode data with a maximum of 8,000 characters

17. **Text** – Variable-length non-Unicode data with a maximum length of \(2^{31} - 1\) (2,147,483,647) characters

18. **Binary** – Fixed-length binary data with a maximum length of 8,000 bytes
19. **Varbinary** – Variable-length binary data with a maximum length of 8,000 bytes

20. **Image** – Variable-length binary data with a maximum length of $2^{31} - 1$ (2,147,483,647) bytes

iii) **Optional Column Constraints**

The Optional Column Constraints are NULL, NOT NULL, UNIQUE, PRIMARY KEY and DEFAULT, used to initialize a value for a new record. The column constraint NULL indicates that null values are allowed, which means that a row can be created without a value for this column. The column constraint NOT NULL indicates that a value must be supplied when a new row is created.

To illustrate, we will use the SQL statement `CREATE TABLE EMPLOYEES` to create the students table with 6 attributes or fields.

```sql
USE university
CREATE TABLE Students
(
    StudentRegNo CHAR(10) NOT NULL UNIQUE,
    StudentFName CHAR(30) NOT NULL,
    StudentMName CHAR(25) NOT NULL,
    StudentLName CHAR(25) NOT NULL,
    YearEnrolled DATE NOT NULL,
    BirthDate DATE NOT NULL,
    CONSTRAINT students_PK PRIMARY KEY (StudentRegNo)
);
```

The first field is ‘StudentRegNo’ with a field type of CHAR. For this field, the field length is 10 characters, and the user cannot leave this field empty (NOT NULL).

Similarly, the second field is StudentFName with a field type CHAR of length 30. After all the table columns are defined, a table constraint, identified by the word CONSTRAINT, is used to create the primary key:
CONSTRAINT StudentsPK PRIMARY KEY(StudentRegNo)

We will discuss the constraint property further later in this unit. Likewise, we can create two more tables a ‘course’ table and ‘AcademicProgram’ table using the CREATE TABLE SQL DDL command as shown in the below example.

```
USE university
CREATE TABLE COURSE
(
    CourseTitle Char(35)  NOT NULL,
    CourseCode Char(30)  NOT NULL,
    CourseUnitsInt  NOT NULL,
    CONSTRAINT COURSE_PK PRIMARY KEY(CourseCode)
);
```

In this example, an ‘AcademicProgram’ table is created with three fields: ProgramID, ProgramName and CreditPoints.

```
USE university
CREATE TABLE AcademicPrograms
(
    ProgramIDInt  NOT NULL IDENTITY,
    ProgramNameChar(50) NOT NULL,
    CreditPointsInt  NOT NULL,
    CONSTRAINT AcademicPrograms_PK PRIMARY KEY(ProgramID)
);
```

You can use ALTER TABLE statements to add and drop constraints. ALTER TABLE allows columns to be removed. When a constraint is added, all existing data are verified for violations.

```
ALTER TABLE table_name
ADD column_name datatype
```

In the example below, we will use the ALTER TABLE statement to the alter table AcademicProgram by adding another column/field called ‘YearCommenced”.

```
```
USE university
GO
ALTER TABLE AcademicProgram
ADD YearCommencedDateTime NOT NULL,

The above SQL ALTER TABLE example will add a column called YearCommenced to the AcademicProgram table.

We can also use alter table use the ALTER TABLE statement to modify a column in a table by using the following SQL statement;

ALTER TABLE table_name
MODIFY column_name column_type

Look at the following example below where we will modify the column ProgramName by changing its data type.

ALTER TABLE AcademicProgram
MODIFY ProgramName VARCHAR(100) NOT NULL;

We can confirm these changes by using SQL statement DESC table_name to verify the data structure changes.

DESC AcademicProgram;

11.2.4 DROP TABLE
The DROP TABLE will remove a table from the database. Make sure you have the correct database selected.

DROP TABLE table_name

For example; if we want to drop/delete table AcademicProgram, we write the SQL statement as follows;

DROP TABLE AcademicProgram

Executing the above SQL DROP TABLE statement will remove the table AcademicProgram from the database university.
11.3 Table Constraints

Table constraints are identified by the CONSTRAINT keyword and can be used to implement various constraints described below.

11.3.1 IDENTITY constraint

We can use the optional column constraint IDENTITY to provide a unique, incremental value for that column. Identity columns are often used with the PRIMARY KEY constraints to serve as the unique row identifier for the table. The IDENTITY property can be assigned to a column with a tinyint, smallint, int, decimal or numeric data type. This constraint:

- Generates sequential numbers
- Does not enforce entity integrity
- Only one column can have the IDENTITY property
- Must be defined as an integer, numeric or decimal data type
- Cannot update a column with the IDENTITY property
- Cannot contain NULL values
- Cannot bind defaults and default constraints to the column

For IDENTITY [(seed, increment)]

- Seed – the initial value of the identity column
- Increment – the value to add to the last increment column

We will use another database example to further illustrate the SQL DDL statements by creating the table tblHotel in this HOTEL database.

```sql
CREATE TABLE  tblHotel
(
    HotelNoInt IDENTITY (1,1),
    Name Char(50) NOT NULL,
    Address Char(50) NULL,
    City Char(25) NULL,
)```
11.3.2 UNIQUE constraint
The UNIQUE constraint prevents duplicate values from being entered into a column.

- Both PK and UNIQUE constraints are used to enforce entity integrity.
- Multiple UNIQUE constraints can be defined for a table.
- When a UNIQUE constraint is added to an existing table, the existing data is always validated.
- A UNIQUE constraint can be placed on columns that accept nulls. Only one row can be NULL.
- A UNIQUE constraint automatically creates a unique index on the selected column.

This is the general syntax for the UNIQUE constraint:

```
[CONSTRAINT constraint_name]
UNIQUE [CLUSTERED | NONCLUSTERED]
(col_name [, col_name2 [..., col_name16]])
[ON segment_name]
```

This is an example using the UNIQUE constraint.
```
CREATE TABLE Students
(
  StudentRegNo CHAR(10) NOT NULL UNIQUE,
)
```

11.3.3 FOREIGN KEY Constraint
The FOREIGN KEY (FK) constraint defines a column, or combination of columns, whose values match the PRIMARY KEY (PK) of another table.

- Values in an FK are automatically updated when the PK values in the associated table are updated/changed.
- FK constraints must reference PK or the UNIQUE constraint of another table.
- The number of columns for FK must be same as PK or UNIQUE constraint.
• If the WITH NOCHECK option is used, the FK constraint will not validate existing data in a table.

• No index is created on the columns that participate in an FK constraint.

This is the general syntax for the FOREIGN KEY constraint:

```
[CONSTRAINT constraint_name]
[FOREIGN KEY (col_name [, col_name2 [,...,
   col_name16]])]
REFERENCES [owner.]ref_table [(ref_col [,,
   ref_col2 [,..., ref_col16]])]
```

In this example, the field HotelNo in the tblRoom table is a FK to the field HotelNo in the tblHotel table shown previously.

```
USE HOTEL
GO
CREATE TABLE  tblRoom
(
    HotelNoInt NOT NULL ,
    RoomNo  Int NOT NULL,
    Type Char(50) NULL,
    Price Money NULL,
    PRIMARY KEY (HotelNo, RoomNo),
    FOREIGN KEY (HotelNo) REFERENCES tblHotel
)
```

11.3.4 CHECK constraint

The CHECK constraint restricts values that can be entered into a table.

• It can contain search conditions similar to a WHERE clause.

• It can reference columns in the same table.

• The data validation rule for a CHECK constraint must evaluate to a Boolean expression.

• It can be defined for a column that has a rule bound to it.

This is the general syntax for the CHECK constraint:

```
[CONSTRAINT constraint_name]
CHECK [NOT FOR REPLICATION] (expression)
```

In this example, the Type field is restricted to have only the types ‘Single’, ‘Double’, ‘Suite’ or ‘Executive’.
USE HOTEL
GO
CREATE TABLE tblRoom
(
    HotelNoInt NOT NULL,
    RoomNoInt NOT NULL,
    Type Char(50) NULL,
    Price Money NULL,
    PRIMARY KEY (HotelNo, RoomNo),
    FOREIGN KEY (HotelNo) REFERENCES tblHotel
    CONSTRAINT Valid_Type
    CHECK (Type IN ('Single', 'Double', 'Suite', 'Executive'))
)

In this second example, the employee hire date should be before January 1, 2004, or have a salary limit of $300,000.

GO
CREATE TABLE SALESREPS
(
    Empl_numInt Not Null
    CHECK (Empl_num BETWEEN 101 and 199),
    Name Char (15),
    AgeInt CHECK (Age >= 21),
    Quota Money CHECK (Quota >= 0.0),
    HireDateDateTime,
    CONSTRAINT QuotaCap CHECK ((HireDate< "01-01-2004") OR (Quota <= 300000))
)

11.3.5 DEFAULT constraint

The DEFAULT constraint is used to supply a value that is automatically added for a column if the user does not supply one.

- A column can have only one DEFAULT.
- The DEFAULT constraint cannot be used on columns with a timestamp data type or identity property.
- DEFAULT constraints are automatically bound to a column when they are created.

The general syntax for the DEFAULT constraint is:

[CONSTRAINT constraint_name]
DEFAULT {constant_expression | niladic-function | NULL}
[FOR col_name]
This example sets the default for the city field to ‘Vancouver’.

```sql
USE HOTEL
ALTER TABLE tblHotel
Add CONSTRAINT df_city DEFAULT 'Vancouver'
FOR City
```

### 11.4 User Defined Types

User defined types are always based on system-supplied data type. They can enforce data integrity and they allow nulls. To create a user-defined data type in SQL Server, choose types under “Programmability” in your database. Next, right click and choose ‘New’ -> ‘User-defined data type’ or execute the `sp_addtype` system stored procedure. After this, type:

```sql
sp_addtypessn, 'varchar(11)', 'NOT NULL'
```

This will add a new user-defined data type called SIN with nine characters. In this example, the field EmployeeSIN uses the user-defined data type SIN.

```sql
CREATE TABLE SINTable
(
EmployeeID INT Primary Key,
EmployeeSIN SIN,
CONSTRAINT CheckSIN
CHECK (EmployeeSIN LIKE
' [0-9][0-9][0-9] – [0-9][0-9] [0-9] – [0-9][0-9] [0-9] – [0-9][0-9] [0-9]'
)
)
Video lecture:

Introduction to SQL - https://tinyurl.com/zlh99lo

DDL Commands - https://tinyurl.com/j992obd

CREATE TABLE Command and Table Constraints: https://tinyurl.com/jegmsh3

INSERT COMMAND: https://tinyurl.com/zn4ko8m
Activity

Activity 11.1
Introduction to SQL Exercises

Aim: To become conversant with basic SQL operations.

Resources: Unit 11 Learning materials and MySQLDBMS or its equivalent.

What to do:

Create a database called ‘Emporium’ with the attached structure below in a ERD and populate it with tanning-related data. UNIQUE constraints were used to guarantee unique values for the identifiers from the entity-relationship model. The BED table used an additional UNIQUE constraint on the drummer_id column to make sure that more than one bed can't be named for the same drummer. All currency columns used the FLOAT datatype.

Duration: Expect to spend about 2 hours on this activity.

Feedback: This activity should be submitted to the course Instructor for an assessment and feedback.
Activity 11.2
Introduction to SQL Exercises

Aim: To become conversant with basic SQL DDL operations.

Resources: Unit 11 Learning materials and MySQL DBMS or its equivalent.

What to do:
Populate the table (MODEL) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

MODEL table

<table>
<thead>
<tr>
<th>model_id</th>
<th>model_name</th>
<th>manufacturer</th>
<th>cost_per_use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Super Crisp</td>
<td>CrispTanCorp</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Super Crisp Deluxe</td>
<td>CrispTanCorp</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Roentgen Radiator</td>
<td>CrispTanCorp</td>
<td>14.5</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 1 hour on this activity

Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 11.3
Introduction to SQL Exercises
Aim: To become conversant with basic SQL operations.
Resources: Unit 11 Learning materials and MySQL DBMS or its equivalent.

What to do:
Populate the table (DRUMMER) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

DRUMMER table

<table>
<thead>
<tr>
<th>drummer_id</th>
<th>drummer_name</th>
<th>band</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Keith Moon</td>
<td>Who, The</td>
</tr>
<tr>
<td>2</td>
<td>John Bonham</td>
<td>Led Zeppelin</td>
</tr>
<tr>
<td>3</td>
<td>Charlie Watts</td>
<td>Rolling Stones, The</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 2 hours on this activity.
Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 11.4
Introduction to SQL Exercises
Aim: To become conversant with basic SQL operations.
Resources: Unit 11 Learning materials and MySQLDBMS or its equivalent.

What to do:
Populate the table (PLAN) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

<table>
<thead>
<tr>
<th>plan_id</th>
<th>plan_name</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walk-In</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Lobster</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Bronze</td>
<td>10</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 30 minutes on this activity
Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 11.5
Introduction to SQL Exercises

Aim: To become conversant with basic SQL operations.

Resources: Unit 11 Learning materials and MySQL DBMS or its equivalent.

What to do:
Populate the table (MEMBERSHIP) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

MEMBERSHIP table

<table>
<thead>
<tr>
<th>plan_id</th>
<th>drummer_id</th>
<th>membership_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1976-05-02</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1976-05-14</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1976-05-28</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1976-06-15</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 2 hours on this activity.
Feedback: This activity should be submitted to the course Instructor for a feedback.
Activity 11.6
Introduction to SQL Exercises
Aim: To become conversant with basic SQL operations.
Resources: Unit 11 Learning materials and MySQLDBMS or its equivalent.

What to do:
Populate the table (BED) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

BED table

<table>
<thead>
<tr>
<th>bed_id</th>
<th>drummer_id</th>
<th>model_id</th>
<th>letter_code</th>
<th>surcharge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>1</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>B</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>NULL</td>
<td>1</td>
<td>C</td>
<td>0</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 2 hours on this activity.
Feedback: This activity should be submitted to the course Instructor for a feedback.
Activity 11.7
Introduction to SQL Exercises

Aim: To become conversant with basic SQL operations.

Resources: Unit 11 Learning materials and MySQLDBMS or its equivalent.

What to do:
Populate the table (BED) with data as detailed below. Please pay attention to the order in which your tables are populated: primary key columns must be populated before dependent foreign key columns.

VISIT table

<table>
<thead>
<tr>
<th>visit_id</th>
<th>plan_id</th>
<th>drummer_id</th>
<th>bed_id</th>
<th>visit_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1976-05-02</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1976-05-10</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1976-05-14</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1976-05-28</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1976-06-15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1976-06-15</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 1 hour on this activity
Feedback: This activity should be submitted to the course Instructor for a feedback.
Unit summary

In this unit you learned about SQL which is divided into two main categories; DDL and DML. These categories show how to create a database and tables in that particular database. For this unit, the emphasis was on data definition language by using SQL where we had a chance of creating database and tables; within their respective data types and required constraints. In summary, this unit has covered how to define structure of a database and its associated tables and how to manipulate the data within.

Assignment

Aim: This assignment is aimed at assessing student knowledge on SQL operation skills

Duration: Expect to spend 2 hours for this activity.

Task 1
Create the database named ‘Organization’ then a table ‘Employee’ with the following attributes and its corresponding data declaration:-

<table>
<thead>
<tr>
<th>ATTRIBUTE (FIELD) NAME</th>
<th>DATA DECLARATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP_NUM</td>
<td>CHAR(3)</td>
</tr>
<tr>
<td>EMP_LNAME</td>
<td>VARCHAR(15)</td>
</tr>
<tr>
<td>EMP_FNAME</td>
<td>VARCHAR(15)</td>
</tr>
<tr>
<td>EMP_INITIAL</td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>EMP_HIREDATE</td>
<td>DATE</td>
</tr>
<tr>
<td>JOB_CODE</td>
<td>CHAR(3)</td>
</tr>
</tbody>
</table>

Task 2
Populate the table created in task 1 with the following data.

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>EMP_LNAME</th>
<th>EMP_FNAME</th>
<th>EMP_INITIAL</th>
<th>EMP_HIREDATE</th>
<th>JOB_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>News</td>
<td>John</td>
<td>G</td>
<td>08-Nov-80</td>
<td>502</td>
</tr>
<tr>
<td>102</td>
<td>Senior</td>
<td>David</td>
<td>H</td>
<td>12-Jul-88</td>
<td>501</td>
</tr>
<tr>
<td>103</td>
<td>Arbough</td>
<td>June</td>
<td>E</td>
<td>01-Dec-96</td>
<td>500</td>
</tr>
<tr>
<td>104</td>
<td>Ramorae</td>
<td>Anne</td>
<td>K</td>
<td>15-Nov-87</td>
<td>501</td>
</tr>
<tr>
<td>105</td>
<td>Johnson</td>
<td>Alice</td>
<td>K</td>
<td>01-Feb-93</td>
<td>502</td>
</tr>
<tr>
<td>106</td>
<td>Smithfield</td>
<td>William</td>
<td></td>
<td>22-Jun-04</td>
<td>500</td>
</tr>
<tr>
<td>107</td>
<td>Alonso</td>
<td>Maria</td>
<td>D</td>
<td>10-Oct-93</td>
<td>500</td>
</tr>
<tr>
<td>108</td>
<td>vWashington</td>
<td>Ralph</td>
<td>B</td>
<td>22-Aug-91</td>
<td>501</td>
</tr>
<tr>
<td>109</td>
<td>Smith</td>
<td>Larry</td>
<td>W</td>
<td>18-Jul-97</td>
<td>501</td>
</tr>
</tbody>
</table>
Task 3
Write the SQL code to change the job code to 501 for the person whose personnel number is 107. After you have completed the task, examine the results, and then reset the job code to its original value.

Task 4
Assuming that the data shown in the Employee table have been entered, write the SQL code that lists all attributes for a job code of 502.

Task 5
Write the SQL code to delete the row for the person named William Smithfield, who was hired on June 22, 2004, and whose job code classification is 500. (Hint: Use logical operators to include all the information given in this problem.)

Task 6
Add the attributes EMP_PCT and PROJ_NUM to the Employee table. The EMP_PCT is the bonus percentage to be paid to each employee.

Task 7
Using a single command, write the SQL code that will enter the project number (PROJ_NUM) = 18 for all employees whose job classification (JOB_CODE) is 500.

Task 7
Using a single command, write the SQL code that will enter the project number (PROJ_NUM) = 25 for all employees whose job classification (JOB_CODE) is 502 or higher.

Task 8
Write the SQL code that will change the PROJ_NUM to 14 for those employees who were hired before January 1, 1994, and whose job code is at least 501. (You may assume that the table will be restored to its original condition preceding this question.)
References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. SQL Examples in creating Databases and column constraints.
3. Unit Summary
Unit 12

SQL – Data Manipulation Language

Introduction

In unit 11, we were introduced to Data Definition Language (DDL) which is a prerequisite for this unit. In this unit, we have an opportunity to learn about Data Manipulation Language (DML) whereby we will learn how to manipulate (query) a database by using different SQL. Please note that this unit is a practical based unit, where as we go along the content, we need to be hands on with a DBMS of our choice.

Unit Outcomes

Upon completion of this unit you should be able to:

- Construct basic queries using Distinct Clause.
- Construct basic queries using Where Clause.
- Construct basic queries using Sand/Or operator.
- Construct basic queries using Aggregate Functions.
- Construct basic queries using Order By Clause.
- Construct basic queries using Special Operator.

Terminologies

Distinct Clause: SQL operator which is used to return only distinct (different) values.

Where Clause: SQL operator which is used to filter records.

And/Or Operator: SQL operator which is used to filter records on one or more conditions.

SQL Aggregate Function: SQL functions which perform a calculation on a set of values and return a single, or summary, value.
12.1 Data manipulation Language (DML)

The SQL Data Manipulation Language (DML) is used to query and modify database data. In this unit, we will describe how to use the SELECT, INSERT, UPDATE, and DELETE SQL DML command statements, defined below.

- **SELECT** – to query data in the database
- **INSERT** – to insert data into a table
- **UPDATE** – to update data in a table
- **DELETE** – to delete data from a table

In the SQL DML statement, please note that:

- Each clause in a statement should begin on a new line.
- The beginning of each clause should line up with the beginning of other clauses.
- If a clause has several parts, they should appear on separate lines and be indented under the start of the clause to show the relationship.
- Upper case letters are used to represent reserved words.
- Lower case letters are used to represent user-defined words.

12.2 INSERT Statement

Before we can manipulate any data from a database, we need to populate that database with data. Consider the following example below, where we are going to create a table called ‘courses’ and then insert data in it.

Note that, before creating the table you need to select particular database where your table will be created. We are going to use database named
‘university’ from previous unit and a table called courses which was created by suing the following syntax. The following are the steps involved;

(i) Creating the table

Use university

CREATE TABLE courses
(
    CourseCode VARCHAR(40) PRIMARY KEY,
    CourseTitle VARCHAR(255),
    CourseUnits INT NOT NULL,
    CourseCredits INT NOT NULL
);

(ii) Confirming the structure of the table

In order to confirm the structure of your table, type the statement

DESC courses;

The following should be the output;

+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
|---------------+--------------+------+-----+---------+-------+
| CourseCode    | varchar(40)  | NO   | PRI | NULL    |       |
| CourseTitle   | varchar(255) | YES  |     | NULL    |       |
| CourseUnits   | int(11)      | NO   |     | NULL    |       |
| CourseCredits | int(11)      | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+

4 rows in set (0.06 sec)

(iii) Inserting Data into the table ‘courses’

INSERT INTO courses VALUES (“OIT214”, "OO Programming", 3, 30);
INSERT INTO courses VALUES (OIT215, "Data Communication", 3, 30);
INSERT INTO courses VALUES (“OIT216”, "Introduction to E-Business", 2, 10);
INSERT INTO courses VALUES (“OIT217”, "Database Design", 2, 10);
INSERT INTO courses VALUES (“OIT218”, "Computer Security", 1, 5);
INSERT INTO courses VALUES (“OIT219”, "Software Design", 1, 20);
Now, we have learned how to insert/populate data into our tables. Please note, you have to be careful, with SQL syntax while creating and pulating the database. One, spelling mistake, then the statement will not go through.

**12.3 SELECT Statement**

The SELECT statement, or command, allows the user to extract data from tables in a selected database. Please, note the syntax of the SELECT statement below:

- In a circumstance that you only want to select few selected columns from that particular table, the following syntax applies.

  ```sql
  SELECT column_name, column_name
  FROM table_name;
  ```

- In a circumstance that you only want to select all columns from that particular table, the following syntax applies.

  ```sql
  SELECT *
  FROM table_name;
  ```

**Example 1:**

Referring to the table ‘courses’ that we had inserted data into; we are going to have a look at what data the table holds after inserting the data in the previous section.

```sql
SELECT *
FROM courses;
```

The following should be the output;
Example 2:
If we want to select one or more column from a particular table; the following syntax holds;

```
SELECT column_name, column_name
FROM table_name;
```

Following, our table `courses` we had populated with data in the previous section;

```
SELECT CourseTitle
FROM courses;
```

The following should be the result;

```
+----------------------------+
<table>
<thead>
<tr>
<th>CourseTitle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Design</td>
</tr>
<tr>
<td>OO Programming</td>
</tr>
<tr>
<td>Data Communication</td>
</tr>
<tr>
<td>Introduction to E-business</td>
</tr>
<tr>
<td>Database Design</td>
</tr>
<tr>
<td>Computer Security</td>
</tr>
</tbody>
</table>
+----------------------------+
6 rows in set (0.00 sec)
The following sections, we are going to learn more conditions that are used with the SELECT statement in querying the database.

12.4 Where Clause

In this section we are going to learn more on the SELECT statements by filtering the data in the database. The WHERE clause is mainly used to filter records. Its syntax is as follows:

\[
\text{SELECT column\_name, column\_name} \\
\text{FROM table\_name} \\
\text{WHERE column\_name \ operator \ value;}
\]

Consider the following example below, where we will be using our university database and ‘courses’ tables from previous unit and section:

Example 1

If we want to know Course Title courses which have 2 units, the syntax will be as follows:

\[
\text{SELECT CourseTitle from courses} \\
\text{WHERE CourseUnits = 2;}
\]

The following should be the output:

+-----------------------------+
| coursetitle                 |
+-----------------------------+
| Introduction to E-business  |
| Database Design             |
+-----------------------------+
2 rows in set (0.00 sec)

Example 2

If we want know how many courses have 2 or more units, we would use the following syntax:-

\[
\text{SELECT CourseTitle from courses} \\
\text{WHERE CourseUnits >= 2;}
\]

The following should be the output;
12.5 Special Operators

The WHERE clause can be used with the following operators in querying the database as illustrated in the examples above;

<table>
<thead>
<tr>
<th>S/N</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BETWEEN</td>
<td>Between an inclusive range</td>
</tr>
<tr>
<td>2</td>
<td>LIKE</td>
<td>Search for a certain pattern</td>
</tr>
<tr>
<td>3</td>
<td>IN</td>
<td>Specifies multiple possible values for a field</td>
</tr>
<tr>
<td>4</td>
<td>&lt;&gt;</td>
<td>Not equal.</td>
</tr>
<tr>
<td>5</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>6</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>7</td>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>8</td>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
</tbody>
</table>

12.6 AND /OR Operators

AND & Or operators are used with the WHERE clause filter records based on more than one condition. Consider the following example below:
**Example 1: AND Operator**

We are going to select all courses which have 3 unit and 30 credits from our table courses from previous section.

```
SELECT * FROM Courses
WHERE CourseUnits=3
AND CourseCredits=30;
```

The result should be as follows:

```
+------------+--------------------+-------------+---------------+
| CourseCode | CourseTitle        | CourseUnits | CourseCredits |
|------------+--------------------+-------------+---------------|
| OIT214     | OO Programming     | 3           | 30            |
| OIT215     | Data Communication | 3           | 30            |
+------------+--------------------+-------------+---------------+
```

2 rows in set (0.00 sec)

**Example 2: Or Operator**

We are going to select all courses which have 3 unit and 30 credits from our table courses from previous section.

```
SELECT * FROM Courses
WHERE CourseUnits=2
OrCourseCredits=20;
```

The following should be the results:

```
+------------+--------------------------------+-------------+---------------+
| CourseCode | CourseTitle                      | CourseUnits | CourseCredits |
|------------+--------------------------------+-------------+---------------|
| OIT 219    | Software Design                  | 1           | 20            |
| OIT216     | Introduction to E-business       | 2           | 10            |
| OIT217     | Database Design                  | 2           | 10            |
+------------+--------------------------------+-------------+---------------+
```

3 rows in set (0.00 sec)

For more querying of data, AND operator and Or operator can be combined.
12.7 SQL Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single, or summary, value. Table 12.2 lists these functions.

Table 12.2: SQL Aggregate Functions

<table>
<thead>
<tr>
<th>S/N</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AVG</td>
<td>Returns the average of all the values, or only the DISTINCT values, in the expression.</td>
</tr>
<tr>
<td>2</td>
<td>COUNT</td>
<td>Returns the number of non-null values in the expression. When DISTINCT is specified, COUNT finds the number of unique non-null values.</td>
</tr>
<tr>
<td>3</td>
<td>COUNT(*)</td>
<td>Returns the number of rows. COUNT(*) takes no parameters and cannot be used with DISTINCT.</td>
</tr>
<tr>
<td>4</td>
<td>MAX</td>
<td>Returns the maximum value in the expression. MAX can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MAX finds the highest value in the collating sequence. MAX ignores any null values.</td>
</tr>
<tr>
<td>5</td>
<td>MIN</td>
<td>Returns the minimum value in the expression. MIN can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MIN finds the value that is lowest in the sort sequence. MIN ignores any null values.</td>
</tr>
<tr>
<td>6</td>
<td>SUM</td>
<td>Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only.</td>
</tr>
</tbody>
</table>

Look at the examples below, illustrating the aggregate functions:

Example 1: MIN

MIN aggregate function returns the smallest value of the selected column. For example, let us filter records to get the course with the lowest number of credits of all courses in the course table;

```sql
SELECT MIN(CourseCredits) AS 'Minimum Course Credits' from courses;
```
The result should be as follows;

<table>
<thead>
<tr>
<th>Minimum Course Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

1 row in set (0.00 sec)

**Example 2: MAX**

MAX aggregate function returns the largest value of the selected column. For example, let's filter records to get the course with the highest number of credits of all courses in the course table;

```sql
SELECT MAX(CourseCredits) AS 'Maximum Course Credits' 
from courses;
```

The result should be as follows;

<table>
<thead>
<tr>
<th>Maximum Course Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
</tr>
</tbody>
</table>

1 row in set (0.00 sec)

**Example 3: COUNT**

COUNT function returns the number of rows that matches specified criteria. Let's look at the example below from courses table where we are querying the total number of courses.

```sql
SELECT COUNT(*) AS 'Number of Courses' from courses;
```

The result should be as follows;

<table>
<thead>
<tr>
<th>Number of Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

1 row in set (0.00 sec)
Example 4: Avg

Avg function returns the average value of a numeric column. Let's look at the example below from courses table where we are querying the average of course credits in the course table.

```
SELECT Avg(CourseCredits) AS 'Average Course Credits' from courses;
```

The above statement should return the following output;

```
+------------------------+
<table>
<thead>
<tr>
<th>Average Course Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.5000</td>
</tr>
</tbody>
</table>
+------------------------+
1 row in set (0.00 sec)
```

Example 5: Sum

SUM function returns the total sum of a numeric column. Let's look at the example below from courses table where we are calculating the total number of units of course units in the course table.

```
SELECT SUM(CourseUnits) AS 'Total Number of Units' from courses;
```

The result should be as follows;

```
+-----------------------+
<table>
<thead>
<tr>
<th>Total Number of Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
+-----------------------+
1 row in set (0.00 sec)
```

12.8 Order By Clause

You use the ORDER BY clause to sort the records in the resulting list. Use `ASC` to sort the results in ascending order and `DESC` to sort the results in descending order.

Example 1: ASC:

```
select * from courses
ORDER BY CourseCode ASC;
```
The results should be as follows:

<table>
<thead>
<tr>
<th>CourseCode</th>
<th>CourseTitle</th>
<th>CourseUnits</th>
<th>CourseCredits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT214</td>
<td>OO Programming</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>OIT215</td>
<td>Data Communication</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>OIT216</td>
<td>Introduction to E-business</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>OIT218</td>
<td>Computer Security</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>OIT219</td>
<td>Software Design</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

6 rows in set (0.00 sec)

**Example 2: DESC:**

```sql
select * from courses
ORDER BY CourseCode DESC;
```

The results should be as follows:

<table>
<thead>
<tr>
<th>CourseCode</th>
<th>CourseTitle</th>
<th>CourseUnits</th>
<th>CourseCredits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIT219</td>
<td>Software Design</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>OIT218</td>
<td>Computer Security</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>OIT217</td>
<td>Database Design</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>OIT216</td>
<td>Introduction to E-business</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>OIT215</td>
<td>Data Communication</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>OIT214</td>
<td>OO Programming</td>
<td>3</td>
<td>30</td>
</tr>
</tbody>
</table>

6 rows in set (0.00 sec)

**12.9 Group By Clause**

The `GROUP BY` clause is used to create one output row per each group and produces summary values for the selected columns, as shown below.
Example:

Using our courses table, let’s select courses whereby we will group our courses by course units.

```sql
select coursetitle AS 'Course Title',
       CourseUnits AS 'Course Units'
from courses GROUP BY courseunits;
```

<table>
<thead>
<tr>
<th>Course Title</th>
<th>Course Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Security</td>
<td>1</td>
</tr>
<tr>
<td>Introduction to E-business</td>
<td>2</td>
</tr>
<tr>
<td>OO Programming</td>
<td>3</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)

By using activities below, we should be more familiar with these SQL functions.

Video lecture:

Special Operators: [https://tinyurl.com/zmr4fje](https://tinyurl.com/zmr4fje)

Aggregate functions: [https://tinyurl.com/z62dfdk](https://tinyurl.com/z62dfdk)
Activity

Activity 12.1
Aim: To acquire practical skills in using basic SQL operations.

Resources: Unit 12 learning materials and MySQL DBMS or equivalent software.

What to do:
Create a database for the fictitious ‘Diva Prop Rentals Company’, which rents WWII-vintage fighter aircraft to a select group of female singing clients: divas. The divas rent aircraft and fly between a numbers of European capital cities and are charged by the company based upon the model of aircraft and the length of the rental. The name of the database should be ‘DivaProp’.

Duration: Expect to spend about 2 hours on this activity.
Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 12.2
Aim: To acquire practical skills in using basic SQL operations.
Resources: Unit 12 learning materials and MySQL Database Software or similar DBMS.

What to do:
Populate the following table which was created in a database ‘Diva Prop Rentals Company’, in activity 12.1.

**AIRCRAFT Table**

<table>
<thead>
<tr>
<th>AIRCRAFT_ID</th>
<th>MODEL</th>
<th>SEATING</th>
<th>FIXED CHARGE</th>
<th>DAILY_RATE_EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Catalina</td>
<td>7</td>
<td>257500</td>
<td>45575</td>
</tr>
<tr>
<td>2</td>
<td>Corsair</td>
<td>1</td>
<td>275050</td>
<td>48280</td>
</tr>
<tr>
<td>3</td>
<td>Hurricane</td>
<td>1</td>
<td>199595</td>
<td>36050</td>
</tr>
<tr>
<td>4</td>
<td>Messerschmitt</td>
<td>1</td>
<td>120000</td>
<td>37595</td>
</tr>
<tr>
<td>5</td>
<td>Mustang</td>
<td>1</td>
<td>223700</td>
<td>39425</td>
</tr>
<tr>
<td>6</td>
<td>Stuka</td>
<td>2</td>
<td>149595</td>
<td>41000</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 1 hour on this activity.
Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 12.3

Aim: To acquire practical skills in using basic SQL operations.

Resources: Unit 12 learning materials and MySQL Database Software or similar DBMS.

What to do:
Populate the following table which was created in a database ‘Diva Prop Rentals Company’, in activity 12.1.

**AIRPORT Table**

<table>
<thead>
<tr>
<th>AIRPORT_ID</th>
<th>CITY</th>
<th>LANDING_FEE_EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Berlin</td>
<td>301000</td>
</tr>
<tr>
<td>2</td>
<td>London</td>
<td>359525</td>
</tr>
<tr>
<td>3</td>
<td>Madrid</td>
<td>150010</td>
</tr>
<tr>
<td>4</td>
<td>Paris</td>
<td>258600</td>
</tr>
<tr>
<td>5</td>
<td>Rome</td>
<td>422840</td>
</tr>
<tr>
<td>6</td>
<td>Warsaw</td>
<td>96000</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 30 minutes on this activity

Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
Activity 12.4

Aim: To acquire practical skills in using basic SQL operations.

Resources: Unit 12 learning materials and MySQL Database Software or similar DBMS.

What to do:

Populate the following table which was created in a database ‘Diva Prop Rentals Company’, in activity 12.1.

**DIVA Table**

<table>
<thead>
<tr>
<th>Diva_Id</th>
<th>Name</th>
<th>BirthDate</th>
<th>Favourite_Aircraft_Id</th>
<th>Discount_pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Barbra</td>
<td>1942-04-24</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Britney</td>
<td>1981-12-02</td>
<td>NULL</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Cher</td>
<td>1946-05-20</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Celine</td>
<td>1968-03-30</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Madonna</td>
<td>1958-08-16</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Maria</td>
<td>1970-03-27</td>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>ZsaZsa</td>
<td>1918-02-06</td>
<td>NULL</td>
<td>2</td>
</tr>
</tbody>
</table>

Duration: Expect to spend about 1 hour on this activity

Feedback: This activity should be submitted to the course Instructor for assessment and feedback.
**Activity 12.5**

**Aim:** To acquire practical skills in using basic SQL operations.

**Resources:** Unit 12 learning materials and MySQL Database Software or similar DBMS.

**What to do:**
Populate the following table which was created in a database ‘Diva Prop Rentals Company’, in activity 12.1.

**DIVA_RENTAL Table**

<table>
<thead>
<tr>
<th>DivaId</th>
<th>Rental_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

**Duration:** Expect to spend about 3 hour on this activity  
**Feedback:** This activity should be submitted to the course Instructor for assessment and feedback.
**Activity 12.6**  
**Aim:** To acquire practical skills in using basic SQL operations.  
**Resources:** Unit 12 learning materials and MySQL Database Software or similar DBMS.

**What to do:**  
Populate the following table which was created in a database ‘Diva Prop Rentals Company’, in activity 12.1.

<table>
<thead>
<tr>
<th>Renta_Id</th>
<th>Depart_Date</th>
<th>Arrival_Date</th>
<th>Aircraft_Id</th>
<th>Deeparture_Airport_Id</th>
<th>Aircraft_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2006-07-18</td>
<td>2006-07-21</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2006-07-20</td>
<td>2006-07-20</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2006-07-25</td>
<td>2006-08-02</td>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2006-07-31</td>
<td>2006-07-31</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2006-08-02</td>
<td>2006-08-04</td>
<td>6</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2006-08-15</td>
<td>2006-08-15</td>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2006-08-07</td>
<td>2006-08-16</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>2006-08-12</td>
<td>2006-08-15</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>2006-08-10</td>
<td>2006-08-11</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>2006-08-17</td>
<td>2006-08-19</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>2006-08-21</td>
<td>2006-08-21</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>2006-08-22</td>
<td>2006-09-02</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Duration:** Expect to spend about 2 hours on this activity  
**Feedback:** This activity should be submitted to the course Instructor for assessment and feedback.
Unit summary

In this unit we learned about all the basic SQL data manipulation commands. This was mainly hands-on based unit where we have learned how to use different SQL operators like WHERE clause, Group By, order By, Aggregate functions and special operators. All these functions are aimed at helping the database end user to filter their data as per their specific requirements.
Assignment

**Aim:** This is assignment is aimed at examining learners knowledge’s of SQL basic ‘select statements. Students have to get the exact answer which will prove that they have done the activity section correctly.

**Duration:** Expect to spend 2hours on this activity.

By using ‘DivaProp’ database in the activity section please answer the following questions;

1. What is the average aircraft fixed charge for all of the models of aircraft?

   **Answer:** €2,042.40

2. What is the average aircraft fixed charge for all of the rentals?

   **Answer:** €2,053.56

3. What price would each diva, including discounts, of a 3 day rental of the Mustang? Sort the results in descending order by price. *(Hint: you'll need to use both the AIRCRAFT and DIVA table in your query, but you do not INNER JOIN them together.)*

   **Answer:** 7 prices ranging from €3419.75 for Mariah down to €2906.79 for Barbra

4. What is the average price a diva would pay, including discounts, of a 5 day rental of the Hurricane?

   **Answer:** €3,548.84

5. What is the total landing fee incurred for any rentals of aircraft that have the letter "c" (uppercase or lowercase) in the model name?

   **Answer:** €15,422.90

6. Excluding diva discounts, what is the total charge (fixed and daily rate) and landing fee for each of the rentals? *(Hint: calculate the number of days for each rental by subtracting the departure date from the arrival date and then adding 1.)*

   **Answer:** 12 rows, total charge ranges from €1,905.95 to €7,578.50 and landing fee ranges from €960.00 to €4,228.40

7. How many times has each model of aircraft been rented? Sort the
results so that the most-often-rented model appears first.

**Answer:** 6 rows, the Stuka has been rented 4 times

8. How many times has each airport been visited either as a departure or arrival city? Sort the results so that the most-often-visited airport appears first. (*Hint: use an OR in the join condition between AIRPORT and RENTAL.*)

**Answer:** 6 cities, Warsaw has been visited 5 times

9. What is the total amount charged per aircraft model (fixed charge and daily rate), excluding diva discounts, for all of the rentals? Sort the results so that the model producing the most income appears first. (*Hint: starting with your query from exercise #6 makes this a snap.*)

**Answer:** 6 models, total charge ranges from €16,233.80 to €2,356.45

10. What is the average profit or loss per aircraft model (total charge minus landing fees), excluding diva discounts, for all of the rentals? Sort the results so that the least profitable model appears first.

**Answer:** 6 models, average profit ranges from €-1,597.15 to €2,811.70

11. On what dates have airplanes departed from London?  
**Hint:** (2 dates)

12. Which divas have favorite airplanes? And what is the model name and seating capacity of those aircraft?  
**Hint:** (4 divas)

13. Which divas have flown to Berlin (arrivals only)? And on which dates did they arrive?  
**Hint:** (2 divas)

14. Which models of aircraft (together with their seating capacities) have visited Paris (arrivals or departures)?  
**Hint:** (3 models, but one visited Paris twice)

15. Which divas have flown in an aircraft that is not their favorite model, other than the Catalina (whether or not they have a favorite)? Display both the diva name and model in your result set.

16. First try to answer the question without the "favorite model" restriction. That is, start off by just finding the aircraft models in which each diva has flown.
Hint: (4 different divas in 4 different models, 6 rows in total)

References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book *Database Design* by Adrienne Watt and Nelson Eng licensed under *Creative Commons Attribution 4.0 International License*.

Download this book for free at [http://open.bccampus.ca](http://open.bccampus.ca)

The following material was written by Grace Mbwete:

1. Introduction
2. All SQL examples in the unit for manipulating data in the Databases.
3. Unit Summary
Unit 13

SQL – Join Statements

Introduction

Programmers frequently join data from a number of different tables in order to obtain more information. In this unit, we will learn about SQL Joins, which allow us to create complex queries, combine data from different tables, and obtain a new result set that can provide us with a better understanding of the data and maximize database flexibility.

Unit Outcomes

Upon completion of this unit you should be able to:

- Explain the different methods for joining tables.
- Construct advanced queries of two or more tables using join operations.

Terminologies

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Join</td>
<td>SQL operator which connects two tables on a column with the same data type.</td>
</tr>
<tr>
<td>Left Outer Join</td>
<td>SQL operator which specifies that all left outer rows be returned.</td>
</tr>
<tr>
<td>Right Outer Join</td>
<td>SQL operator which includes, in its result set, all rows from the right table that did not meet the condition specified.</td>
</tr>
<tr>
<td>Group By</td>
<td>Aggregate functions in the SELECT clause &lt;select&gt; list provide information about each group instead of individual rows.</td>
</tr>
<tr>
<td>Inner Join</td>
<td>SQL operator which connects two tables on a column with the same data type.</td>
</tr>
<tr>
<td>Left Outer Join</td>
<td>SQL operator which specifies that all left outer rows be returned.</td>
</tr>
</tbody>
</table>
13.1 SQL Join Statements

Joining two or more tables is the process of comparing the data in specified columns and using the comparison results to form a new table from the rows that qualify. A join statement:

- Specifies a column from each table
- Compares the values in those columns row by row
- Combines rows with qualifying values into a new row

Although the comparison is usually for equality – values that match exactly – other types of joins can also be specified. All the different joins such as inner, left (outer), right (outer), and cross join will be described below.

13.2 Inner Joins

An inner join connects two tables on a column with the same data type. Only the rows where the column values match are returned; unmatched rows are discarded. The following is the syntax which illustrate how inner join operator works:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

13.3 Outer Joins

13.3.1 Left Outer Join

A left outer join specifies that all left outer rows be returned. All rows from the left table that did not meet the condition specified are included in the results set, and output columns from the other table are set to NULL.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```
13.3.2 Right Outer Joins

A right outer join includes, in its result set, all rows from the right table that did not meet the condition specified. Output columns that correspond to the other table are set to NULL. Below is an example using the new syntax for a right outer join.

```sql
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

The group By Clause

Video lecture:

[https://tinyurl.com/zlpvaom](https://tinyurl.com/zlpvaom)
## Activity

**Activity 13.1**

**Aim:** To acquire practical skills in using basic SQL Join operations.

**Resources:** Unit 12 & 13 learning materials and MySQL Database Software or similar DBMS.

**What to do:**

By using the database created in unit 12; please attempt the following:-

1. Show the list of all drummers, and the code of the tanning bed that has been named for each, or "N/A" if a bed has not been named for the drummer.

   **Hint:** 7 drummers, 3 have not had beds named after them

2. Show the list of all tanning beds, and the name of the drummer for which each has been named, or just the letter code again for unnamed beds.

   **Hint:** 6 beds, two have not been named

3. What has each drummer paid (rate + surcharge) for their visits, broken down by the bed they used? If a drummer uses a bed named for himself, he does not pay the surcharge. *(Hint: You'll need an extra copy of BED. Use COALESCE() to "refund" the surcharge. You may run into trouble using ROUND(), so skip it.)*

   **Hint:** 16 drummer and bed combinations, Charlie pays $67.25 for B, Keith pays $45.00 for D

**How to do it:**

By using ‘Emporium’ database created in unit 11; attempt the above referred tasks.

**Duration:** Expect to spend about 1 hour on this activity

**Feedback:** This activity should be submitted to the course Instructor for assessment and feedback.
Unit summary

In this unit you learned about inner and outer join function which connects two tables in which it allows creation of a new table with comparison of rows that have qualified.

Case Study

Aim: This activity is a continuation of unit 11 and 12; where you have to be competent with the basic SQL functions and complement them with OUTER and INNER join statements.

By using ‘DivaProp’ database created in unit 12; attempt the following tasks as part of this unit assessment:-

1. The company institutes a new promotional incentive so that divas that fly on their favorite aircraft get a bonus 5% discount on each rental that applies even if the rental is shared (but that's the maximum discount even if more than one diva fly together in their favorite model). Create a view called INCENTIVE_DISCOUNT that contains the columns rental_id and incentive_factor (0.95) for each of the rentals in the RENTAL table. This step doesn't require an outer join.
   
   i. **Hint: rentals 2, 3, 5, 6, 11**

2. What is the income, expense, discount factor, incentive factor, and profit/loss (including discounts and incentives) for each of the rentals? *(Hint: Use the RENTAL_SUMMARY, RENTAL_DISCOUNT (from last week), and INCENTIVE_DISCOUNT views. Use the COALESCE() function to calculate an incentive factor of 1.00 when it doesn't apply to a particular rental.)*
   
   i. **Hint: 12 rentals, profit ranges from €-1,912.90 €6,466.93**

3. What is the total and average profit/loss for all rentals, including diva discounts and incentives?
   
   i. **Hint: total profit of €13,938.10**
   
   ii. **average per-rental profit of €1,161.51**

4. How many times has each diva flown on a model of aircraft that is not her favorite?
   
   a. *(Hint: first try to answer the question without the bit about it not being her favorite aircraft. No outer join is needed, but you will need to use the COALESCE() function to force the DIVA.favourite_aircraft_id column to be an impossible value when it is NULL (e.g. 0).)*
13 rows; Barbra, Cher, Madonna, and Mariah have each flown twice on the Catalina

References and Further Reading


Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is a derivative copy of materials from the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 International License.

Download this book for free at http://open.bccampus.ca

The following material was written by Grace Mbwete:

1. Introduction
2. Unit Summary
Unit 14

Database Development Process

Introduction

After learning all the basic theories and practical techniques of database design in all previous thirteen units of this course; the learner will now have the required skills of designing and implement a database given user requirements. This unit will cover the process of software development life cycle, database development process which will assist the learner in achieving design and implementation of database for a particular case.

Unit Outcomes

Upon completion of this unit you will be able to:

- Understand the basic concepts of software development life cycle.
- Analyze user requirements for database design and implementation process.
- Convert user requirements into a logical design.
- Realize the logical design and implement the database.
- Test and maintain the implemented database.

Terminologies

- **Analysis**: Starts by considering the statement of requirements and finishes by producing a system specification. Used to confirm the understanding of requirements with the user.
- **Data requirements document**: Begins with a system specification, produces design documents and provides a detailed description of how a system should be constructed.
- **Design**: Involves consultation with, and agreement among, stakeholders as to what they want from a system; expressed as a statement of
14.1 Overview

A core aspect of software engineering is the subdivision of the development process into a series of phases, or steps, each of which focuses on one aspect of the development. The collection of these steps is sometimes referred to as the *software development life cycle (SDLC)*. The software product moves through this life cycle (sometimes repeatedly as it is refined or redeveloped) until it is finally retired from use. Ideally, each phase in the life cycle can be checked for correctness before moving on to the next phase.

14.2 Software Development Life Cycle (SDLC)

Let us start with an overview of the *waterfall model* such as you will find in most software engineering textbooks. This waterfall figure, seen in Figure 14.1, illustrates a general waterfall model that could apply to any computer system development. It shows the process as a strict sequence.
of steps where the output of one step is the input to the next and all of one step has to be completed before moving onto the next.

![Waterfall Model](image)

Figure 14.1: Illustration of waterfall model, by G.Mbwete

Waterfall model is used to illustrate the tasks that are required, together with the input and output for each activity. What is important is the scope of the activities, which can be summarized as follows:

i. *Establishing requirements* involves consultation with, and agreement among, stakeholders about what they want from a system, expressed as a statement of requirements.

ii. *Analysis* starts by considering the statement of requirements and finishes by producing a system specification. The specification is a formal representation of what a system should do, expressed in terms that are independent of how it may be realized.
iii. *Design* begins with a system specification, produces design documents and provides a detailed description of how a system should be constructed.

iv. *Implementation* is the construction of a computer system according to a given design document and taking into account the environment in which the system will be operating (e.g., specific hardware or software available for the development). Implementation may be staged, usually with an initial system that can be validated and tested before a final system is released for use.

v. *Testing* compares the implemented system against the design documents and requirements specification and produces an acceptance report or, more usually, a list of errors and bugs that require a review of the analysis, design and implementation processes to correct (testing is usually the task that leads to the waterfall model iterating through the life cycle).

vi. *Maintenance* involves dealing with changes in the requirements or the implementation environment, bug fixing or porting of the system to new environments (e.g., migrating a system from a standalone PC to a UNIX workstation or a networked environment). Since maintenance involves the analysis of the changes required, design of a solution, implementation and testing of that solution over the lifetime of a maintained software system, the waterfall life cycle will be repeatedly revisited.

### 14.3 Database Life Cycle

We can use the waterfall cycle as the basis for a model of database development that incorporates three assumptions:

i. We can separate the development of a database – that is, specification and creation of a schema to define data in a database – from the user processes that make use of the database.

ii. We can use the three-schema architecture as a basis for distinguishing the activities associated with a schema.
iii. We can represent the constraints to enforce the semantics of the data once within a database, rather than within every user process that uses the data.

![Waterfall Model for Databases](image)

Figure 14.2: Illustration of *waterfall* model for databases, by G.Mbwete

Using these assumptions and Figure 14.2, we can see that this diagram represents a model of the activities and their outputs for database development. It is applicable to any class of DBMS, not just a relational approach. Database application development is the process of obtaining real-world requirements, analyzing requirements, designing the data and functions of the system, and then implementing the operations in the system.

**14.3.1 Requirements Gathering**

The first step is *requirements gathering*. During this step, the database designers have to interview the customers (database users) to understand
the proposed system and obtain and document the data and functional requirements. The result of this step is a document that includes the detailed requirements provided by the users.

Establishing requirements involves consultation with, and agreement among, all the users as to what persistent data they want to store along with an agreement as to the meaning and interpretation of the data elements. The data administrator plays a key role in this process as they overview the business, legal and ethical issues within the organization that impact on the data requirements.

The *data requirements document* is used to confirm the understanding of requirements with users. To make sure that it is easily understood, it should not be overly formal or highly encoded. The document should give a concise summary of all users’ requirements – not just a collection of individuals’ requirements – as the intention is to develop a single shared database.

The requirements should not describe how the data is to be processed, but rather what the data items are, what attributes they have, what constraints apply and the relationships that hold between the data items.

### 14.3.2 Analysis

Data analysis begins with the statement of data requirements and then produces a conceptual data model. The aim of analysis is to obtain a detailed description of the data that will suit user requirements so that both high and low level properties of data and their use are dealt with. These include properties such as the possible range of values that can be permitted for attributes (e.g., in the school database example, the student course code, course title and credit points).

The conceptual data model provides a shared, formal representation of what is being communicated between clients and developers during database development – it is focused on the data in a database, irrespective of the eventual use of that data in user processes or implementation of the data in specific computer environments. Therefore,
a conceptual data model is concerned with the meaning and structure of data, but not with the details affecting how they are implemented.

The conceptual data model then is a formal representation of what data a database should contain and the constraints the data must satisfy. This should be expressed in terms that are independent of how the model may be implemented. As a result, analysis focuses on the questions, “What is required?” not “How is it achieved?”

14.3.4 Logical Design
Database design starts with a conceptual data model and produces a specification of a logical schema; this will determine the specific type of database system (network, relational, object-oriented) that is required. The relational representation is still independent of any specific DBMS; it is another conceptual data model.

We can use a relational representation of the conceptual data model as input to the logical design process. The output of this stage is a detailed relational specification, the logical schema, of all the tables and constraints needed to satisfy the description of the data in the conceptual data model. It is during this design activity that choices are made as to which tables are most appropriate for representing the data in a database. These choices must take into account various design criteria including, for example, flexibility for change, control of duplication and how best to represent the constraints. It is the tables defined by the logical schema that determine what data are stored and how they may be manipulated in the database.

Database designers familiar with relational databases and SQL might be tempted to go directly to implementation after they have produced a conceptual data model. However, such a direct transformation of the relational representation to SQL tables does not necessarily result in a database that has all the desirable properties: completeness, integrity, flexibility, efficiency and usability. A good conceptual data model is an essential first step towards a database with these properties, but that does
not mean that the direct transformation to SQL tables automatically produces a good database. This first step will accurately represent the tables and constraints needed to satisfy the conceptual data model description, and so will satisfy the completeness and integrity requirements, but it may be inflexible or offer poor usability. The first design is then flexed to improve the quality of the database design. *Flexing* is a term that is intended to capture the simultaneous ideas of bending something for a different purpose and weakening aspects of it as it is bent.

Figure 14.3 summarizes the iterative (repeated) steps involved in database design, based on the overview given. Its main purpose is to distinguish the general issue of what tables should be used from the detailed definition of the constituent parts of each table – these tables are considered one at a time, although they are not independent of each other. Each iteration that involves a revision of the tables would lead to a new design; collectively they are usually referred to as second-cut designs, even if the process iterates for more than a single loop.

![Diagram](image-url)
First, for a given conceptual data model, it is not necessary that all the user requirements it represents be satisfied by a single database. There can be various reasons for the development of more than one database, such as the need for independent operation in different locations or departmental control over “their” data. However, if the collection of databases contains duplicated data and users need to access data in more than one database, then there are possible reasons that one database can satisfy multiple requirements, or issues related to data replication and distribution need to be examined.

Second, one of the assumptions about database development is that we can separate the development of a database from the development of user processes that make use of it. This is based on the expectation that, once a database has been implemented, all data required by currently identified user processes have been defined and can be accessed; but we also require flexibility to allow us to meet future requirements changes. In developing a database for some applications, it may be possible to predict the common requests that will be presented to the database and so we can optimize our design for the most common requests.

Third, at a detailed level, many aspects of database design and implementation depend on the particular DBMS being used. If the choice of DBMS is fixed or made prior to the design task, that choice can be used to determine design criteria rather than waiting until implementation. That is, it is possible to incorporate design decisions for a specific DBMS rather than produce a generic design and then tailor it to the DBMS during implementation.

It is not uncommon to find that a single design cannot simultaneously satisfy all the properties of a good database. So it is important that the designer has prioritized these properties (usually using information from the requirements specification); for example, to decide if integrity is more
important than efficiency and whether usability is more important than flexibility in a given development.

At the end of our design stage, the logical schema will be specified by SQL data definition language (DDL) statements, which describe the database that needs to be implemented to meet the user requirements.

### 14.3.5 Implementation

Implementation involves the construction of a database according to the specification of a logical schema. This will include the specification of an appropriate storage schema, security enforcement, external schema and so on. Implementation is heavily influenced by the choice of available DBMSs, database tools and operating environment. There are additional tasks beyond simply creating a database schema and implementing the constraints – data must be entered into the tables, issues relating to the users and user processes need to be addressed, and the management activities associated with wider aspects of corporate data management need to be supported. In keeping with the DBMS approach, we want as many of these concerns as possible to be addressed within the DBMS. We look at some of these concerns briefly now.

In practice, implementation of the logical schema in a given DBMS requires a very detailed knowledge of the specific features and facilities that the DBMS has to offer. In an ideal world, and in keeping with good software engineering practice, the first stage of implementation would involve matching the design requirements with the best available implementing tools and then using those tools for the implementation. In database terms, this might involve choosing vendor products with DBMS and SQL variants most suited to the database we need to implement. However, we don’t live in an ideal world and more often than not, hardware choice and decisions regarding the DBMS will have been made well in advance of consideration of the database design. Consequently, implementation can involve additional flexing of the design to overcome any software or hardware limitations. The following are the stages involved in the implementation of database:

1. **Realizing the Design**
After the logical design has been created, we need our database to be created according to the definitions we have produced. For an implementation with a relational DBMS, this will probably involve the use of SQL to create tables and constraints that satisfy the logical schema description and the choice of appropriate storage schema (if the DBMS permits that level of control).

One way to achieve this is to write the appropriate SQL DDL statements into a file that can be executed by a DBMS so that there is an independent record, a text file, of the SQL statements defining the database. Another method is to work interactively using a database tool like SQL Server Management Studio or Microsoft Access. Whatever mechanism is used to implement the logical schema, the result is that a database, with tables and constraints, is defined but will contain no data for the user processes.

ii) Populating the Database

After a database has been created, there are two ways of populating the tables – either from existing data or through the use of the user applications developed for the database.

For some tables, there may be existing data from another database or data files. For example, in establishing a database for a hospital, you would expect that there are already some records of all the staff that have to be included in the database. Data might also be brought in from an outside agency (address lists are frequently brought in from external companies) or produced during a large data entry task (converting hard-copy manual records into computer files can be done by a data entry agency). In such situations, the simplest approach to populate the database is to use the import and export facilities found in the DBMS.

Facilities to import and export data in various standard formats are usually available (these functions are also known in some systems as loading and unloading data). Importing enables a file of data to be copied directly into a table. When data are held in a file format that is not
appropriate for using the import function, then it is necessary to prepare
an application program that reads in the old data, transforms them as
necessary and then inserts them into the database using SQL code
specifically produced for that purpose. The transfer of large quantities of
existing data into a database is referred to as a bulk load. Bulk loading of
data may involve very large quantities of data being loaded, one table at a
time so you may find that there are DBMS facilities to postpone
constraint checking until the end of the bulk loading.

### iii) Supporting Users and User Processes

Use of a database involves user processes (either application programs or
database tools) which must be developed outside of the database
development. In terms of the three-schema architecture we now need to
address the development of the external schema. This will define the data
accessible to each user process or group of user processes. In reality,
most DBMSs, and SQL itself, do not have many facilities to support the
explicit definition of the external schema. However, by using built-in
queries and procedures, and with appropriate security management, it is
possible to ensure access to data by a user process is limited to a tailored
subset of the entire database content.

In addition to ensuring that appropriate data access for each user process
is achieved, the database developer needs to address other user-related
issues. Examples of such issues include: reporting of error conditions,
constraint enforcement, automated processing using triggers, grouping of
activities into transactions, defining database procedures and functions
and data security (in addition to the general database and user process
access control).

### 14.3.6 Testing

The aim of testing is to uncover errors in the design and implementation
of the database, its structure, constraints and associated user and
management support. Testing is usually considered to involve two main
tasks – validation and verification. Without adequate testing users will
have little confidence in their data processing.
Validation answers the question: has the right database been developed to meet the requirements? It attempts to confirm that the right database has been constructed, with the right characteristics to meet the specified requirements.

Verification answers the question: has the database design been implemented correctly? Verification ensures that the processing steps, constraints and other ‘programmed’ components of the database (security, backup, recovery, audit trails, etc.) have been correctly implemented and contain no errors in program logic or execution sequences.

Of course, testing does not just take place only after all the above development steps are complete. It is usually applied throughout the stages in the development processes and includes appropriate reviews to scrutinise the outputs of the development activities. The aim is to identify errors as soon as possible in the development life cycle.

14.3.7 Maintenance
Databases are one of the more enduring software engineering artefacts; it is not uncommon to find database implementations whose use can be traced back for 15 years or more. Consequently, maintenance of the database is a key issue. Maintenance can take three main forms:

i) Operational maintenance, where the performance of the database is monitored. If it falls below some acceptable standard, then reorganization of the database, usually in the form of adjustments to the storage schema, takes place to ensure that performance is maintained at an acceptable level.

ii) Porting and implementation maintenance, in which the DBMS, the user processes, the underlying computer system or some other aspect undergoes changes that require the database implementation to be revised. Of course, the extreme form of porting occurs when the database is to be entirely replaced – in which case the entire development life cycle is usually followed using the existing database and its limitations as the starting point for requirements analysis. Adjustments to the storage
schema are almost inevitable as new data storage capabilities become available. This can involve both restructuring and reorganization, depending on the scope of the changes taking place.

**iii) Requirements change**, where the original requirement specification changes, usually because databases are frequently used for purposes for which they were not originally designed. This involves restructuring and typically involves a ‘mini life cycle’ related to the development of changes to meet the new requirements.

If we follow the three-schema architecture approach we would hope to minimize the impact of change and build systems which are easily maintained.

---

**Video lecture**

[https://tinyurl.com/juu9f2e](https://tinyurl.com/juu9f2e)
## Activity 14.0
Database Development Process

**Motivation:** To become conversant with basic concepts of database development life cycle.

**Resources:** Internet access and Unit 14 learning materials.

**What to do:** Provide answers for the following questions:

1. Explain in detail, three software development methodologies that would be suitable for databases.
2. What does the acronym SDLC mean, and what does an SDLC portray?
3. What needs to be modified in the waterfall model to accommodate database design?
4. Provide the iterative steps involved in database design.

**Duration:** Expect to spend about 1 hour on this activity

**Feedback:** This activity should be submitted to the course Instructor for assessment and feedback.
Unit summary

In this unit you learned about Database Development Process. In summary, relational database systems underpin the majority of the managed data storage in computer systems. In this unit we have considered database development as an instance of the waterfall model of the software development life cycle. We have seen that the same activities are required to develop and maintain databases that meet user requirements.

Review Questions

Aim: The following review questions are aimed at assessing the students’ knowledge and understanding of software development life cycle and database development process.

1. Explain in details, the meaning of Software Development Life Cycle.

2. Describe the importance of Software Development Life Cycle.

3. What is SDLC model? Mention three types of the most common used SDLC models.

4. What is a waterfall model? Outline the steps in waterfall model.

5. What are the advantages of using waterfall model?

6. Explain when to use waterfall model.

7. What are the disadvantages of using waterfall model?

8. Explain in details the steps involved in Database Life Cycle.

9. Describe the tasks involved during implementation stage of database life cycle.

10. Outline the activities which take place during testing and evaluation of a database.
References and Further Reading:


   Download this book for free at http://open.bccampus.ca


Attribution

The content of this unit (Database Development Process) of the Learning Material – Introduction to Databases (including its images, unless otherwise noted) is from Chapter 13 of the book Database Design by Adrienne Watt and Nelson Eng licensed under Creative Commons Attribution 4.0 License.

Download this book for free at http://open.bccampus.ca

The following material of this unit was written by Grace Mbwete:

1. Introduction
2. Activity 14.0
3. Unit Summary
Video 20 – Review Questions and Activities

https://tinyurl.com/j59x9rz