NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNLOGY

COURSE CODE: CIT 843

COURSE TITLE: INTRODUCTION TO DATABASE MANAGEMENT SYSTEM

COURSE GUIDE

Course Code **CIT 843**

Course Title **Introduction to Database Management System**

Course Developer/Writer

Course Editor

Programme Leader

Course Coordinator

NATIONAL OPEN UNIVERSITY OF NIGERIA

CONTENTS **PAGE**

Introduction

Let me welcome you to this course titled CIT 843 - Database Management System. I promise enjoyment and reward as you study Database Management System.

Database Management System is a two credit uint course available to all students offering Masters of Science (M.Sc.) Computer and Information Technology (CIT).

A Database Management System (DBMS) is a set of software programs that controls the organization, storage, management, and retrieval of data in a database. It is a set of pre-written programs that are used to store, update and retrieve a Database. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system. Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Course Aims

The aim of the course is not complex. The course will allow you to develop background knowledge as well as core expertise in Database Management Systems.

Course Objectives

To achieve the aims set out, the course has a set of objectives. Each unit has specific objectives which are included at the beginning of unit. You should read these objectives before you study the unit. You may wish to refer to them during your study to check on your progress. You should always look at the unit objectives after completion of each unit. By doing so, you would have followed the instructions in the unit.

Below are the comprehensive objectives of the course as a whole. By meeting these objectives, you should have achieved the aims of the course as a whole. In addition to the aims above, this course sets out some objectives. Thus, after going through the course, you should be able to:

- Understand database Design and Normalization techniques.
- Use Structured Query Language and Transaction Management.
- Understand the Importance of backup and recovery techniques.
- Develop database system to handle real world problem.

Working through this Course

To complete this course you are required to read each study unit, read the textbooks and read other materials which may be provided by the National Open University of Nigeria.

Each unit contains self-assessment exercises and at certain points in the course you would be required to submit assignments for assessment purposes. At the end of the course there is a final examination. The course should take you about a total 15 weeks to complete. Below you will find listed all the components of the course, what you have to do and how you should allocate your time to each unit in order to complete the course on time and successfully.

This course entails that you spend a lot of time to read. I would advice that you avail yourself the opportunity of attending the tutorial sessions where you have the opportunity of comparing your knowledge with that of other people.

What you will learn in this Course

This course consists of units and course guide. This course guide tells you briefly what the course is all about, what course materials you will be using and how you can work with these materials. In addition, it advocates some general guidelines for the amount of time you are likely to spend on each unit of the course in order to complete it successfully.

It gives you advice in terms of your Tutor-marked Assignment which will be made available in the assignment file. There will be regular tutorial classes that are related to the course. It is advisable for you to attend these tutorial sessions. The course will prepare you for the challenges you will meet in the database management systems.

The Course Materials

The main components of the course are:

1. The Course Guide
2. Study Units
3. References/Further Readings
4. Assignments
5. Presentation Schedule

Study Units

The study units in this course are as follows:

Module 1 Database Management systems Concepts

Unit 1 the Basic Concepts of DBMS

Unit 2 Data Modeling Overview
Unit 3 Entity-Relationship Modeling
Unit 4 Relational Data Integrity: Conversion of E-R Model to Relational Model
Unit 5 Data Redundancy and Normalization
Unit 6 Relational Algebra

Module 2 Structured Query Languages and Transaction Management

Unit 1 Structured Query Language (SQL)
Unit 2 SQL Functions
Unit 3 Transactions and Concurrency Management
Unit 4 Security
Unit 5 Database Architectures

Module 3 Design and Development of Database Applications

Unit 1 Introduction to Microsoft Access Tables
Unit 2 Introduction to Microsoft Access Queries
Unit 3 Introduction to Microsoft Access Forms
Unit 4 Introduction to Microsoft Access Reports

Textbooks and References

Brainbell.com (2008). Microsoft Access Tutorial. Retrieved June 20th, 2008, from http://www.brainbell.com/tutorials/ms-office/Access_2003/
Bcschool.net (2003-2006). Create Database Applications using Microsoft Access, Retrieved June 20th, 2008, from http://www.bcshool.net/staff/accesshelp.htm
cisnet.baruch.cuny.edu (2008). Microsoft Access Tutorial. Retrieved January 15th, 2008, from http://cisnet.baruch.cuny.edu/holowczak/classes/2200/access/accessall.html
databasedev.co.uk (2003-2006). *Data Redundancy Defined - Relational Database Design.* In, Database Solutions for Microsoft Access, Retrieved October 10th, 2006, from: http://www.databasedev.co.uk/data-redundancy.html
David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Microsoft.com (2009). Microsoft Access help file. Retrieved March 15th, 2009 from http://microsoft.com/office/access/default.htm.
Microsoft.com (2009). Microsoft Access Tutorial: Retrieved March 15th, 2009 from http://www.bcshool.net/staff/accesshelp.htm
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Presentation Schedule

Your course materials have important dates for the early and timely completion and submission of your TMA and attending tutorials. You should remember that you are required to submit all your assignments by the stipulated time and date. You should guide against falling behind in your work.

Assessment

There are three aspects to the assessment of the course. First is made up of self-assessment exercises, second consists of the tutor-marked assignments and third is the written examination/end of course examination.

You are advised to do the exercises. In tackling the assignments, you are expected to apply information, knowledge and techniques you gathered during the course. The assignment must be submitted to your facilitator for formal assessment in accordance with the deadlines stated in the presentation schedule and the assignment file. The work you submit to your tutor for assessment will count for 30% of your total course work. At the end of the course you will need to sit for a final or end of course examination of about three hour duration. This examination will count for 70% of your total course mark.

Assessment File

Assessment file for this course will be made available to you. In this file, you will find details of work that you must submit to your tutor for marking. The marks you obtain in the continuous assessment will count towards your final marks. You are expected to pass both the continuous assessment and the final examination.

Tutor-Marked assignment

The TMA is a continuous assessment component of your course. It accounts for 30% of the total score. You will be given four (4) TMAs to answer. The three of these must be answered before you are allowed to sit for the end of course examination. The TMAs would be given to you by your facilitator and returned after you have done the assignment. Assignment questions for the units in this course are contained in the assignment file. You will be able to complete your assignment from the information and materials contained in your reading, references and study units. However, it is desirable in all degree level of education to demonstrate that you have read and researched more into your references, which will give you a wider view point and may provide you with a deeper understanding of the subject.

Make sure that each assignment reaches your facilitator on or before the deadline given in the presentation schedule and assignment file. If for any reason you can not complete your work on time, contact your facilitator before the assignment is due to discuss the

possibility of an extension. Extension will not be granted after the due date unless there are exceptional circumstances.

Final Examination and Grading

The end of course examination for Introduction to database management Systems will be for about three hours and it has a value of 70% of the total course work. The examination will consists of questions, which will reflect the type of self-testing, practice exercise and tutor-marked assignment problems you have previously encountered. All areas of the course will be assessed.

Kindly use the time between finishing the last unit and sitting for the examination to revise the whole course. You might find it useful to review your self-test, TMA,s and comments on them before the examination. The end of course examination covers information from all parts of the course.

Course Marking Scheme

Assignment	**Marks**
Assignment 1 – 4	Four assignments, best three marks of the four count at 10% each – 30% of the course marks.
End of course examination	70% of overall course marks.
Total	100% of course materials.

Course Overview

The first module unit focuses on the meaning, concepts and advantages of database management system. Module two deals with architecture of the database management system, relational database integrity, transaction and concurrency management, redundancy and associated problems. The third module introduces you to Microsoft Access as an example of database management systems.

How to Get the Most from this Course

Although you will be required to study the units on your own, arrangements have been made for regular interactions with your tutor at the study center. The tutor is expected to conduct tutorials and useful discussion sessions with you and the other members at the study center. Please be available at each tutorial session and participate actively.

Facilitators/Tutors and Tutorials

There are 16 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials as well as the name and phone number of your facilitator, as soon as you are allocated a tutorial group.

Your facilitator will mark and comment on your assignments, keep a close watch on your progress and any difficulty you might face and provide assistance to you during the course. You are expected to mail your Tutor Marked Assignments to your facilitator before the schedule date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not delay to contact your facilitator by telephone or e-mail if you need assistance.

The following might be circumstances in which you would find assistance necessary, hence you would have to contact your facilitator if:

- You do not understand any part of the study or the assigned readings.
- You have difficulty with the self-tests
- You have a question or problem with an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only chance to have face to face contact with your course facilitator and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study.

To gain much benefit from the course tutorials prepare a question list before attending them. You will learn a lot from participating actively in discussions.

Summary

Introduction to Database Management Systems is a course that teaches learners to create and maintain their own database systems using software readily available in the industry such as MS Access. It also provides knowledge on theoretical concepts like relational algebra and query processing.

You will be required to design your own simple information retrieval system for a given application. This will give you thorough exposure to a multitude of DBMS tasks, such as database creation, maintenance, query processing etc. At this stage you would summarize your experiences with the knowledge gained. You would also be made to provide feedback to the course instructor about your view of the pros and cons of a DBMS from your perspective, and about how the course enhanced your sphere of knowledge, and how the course can be improved even further. This would serve the purpose of the course instructor learning from the learners about the application side of things and also about better structuring of courses.
I wish you success in the course and I hope that you will find it both interesting and useful. Thank you.

Course Code **CIT 843**

Course Title **Introduction to Database Management System**

Course Developer/Writer

Course Editor

Programme Leader

Course Coordinator

NATIONAL OPEN UNIVERSITY OF NIGERIA

Module 1: Database Management systems Concepts

1.0 Introduction

Data Management is one of the areas of Computer Science that has applications in almost every field. In this unit, we shall examine some basic terms in database management system.

2.0 Objectives

By the end of this unit, you should be able to:

a. Define database
b. Know why you need database management system
c. Know the advantages of using database management system

3.0 What is Database?

a. A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.

b. A Database (DB) is structure that can store information about:
   i. multiple types of entities;
   ii. the attributes that describe those entities; and
   iii. the relationships among the entities

c. A Database (DB) is collection of related data - with the following properties:
   i. A DB is designed, built and populated with data for a specific purpose
   ii. A DB represents some aspect of the real world.

d. An integrated, self-describing collection of *related* data.

   i. **Integrated:** Data is stored in a uniform way, typically all in one place (a single physical computer for example)
   ii. **Self-Describing:** A database maintains a description of the data it contains (Catalog)
   iii. **Related:** Data has some relationship to other data. In a University we have learners who take *courses* taught by instructors
   iv. By taking advantage of relationships and integration, we can provide *information* to users as opposed to simply *data*.
   v. We can also say that the database is a *model* of what the users perceived.
   vi. Three main categories of models:
      1. **User or Conceptual Models:** How users perceive the world and/or the business.
      2. **Logical Models:** Represent the logic of how a business operates. For example, the relationship between different entities and the flow of data through the organization. Based on the User's model.
      3. **Physical Models:** Represent how the database is actually implemented on a computer system. This is based on the logical model.

3.1 Database Management System

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.

Database Management System (DBMS) is a collection of software programs that are used to define, construct, maintain and manipulate data in a database. Database System (DBS) contains:

a. The Database;
b. The DBMS; and
c. Application Programs (what users interact with)

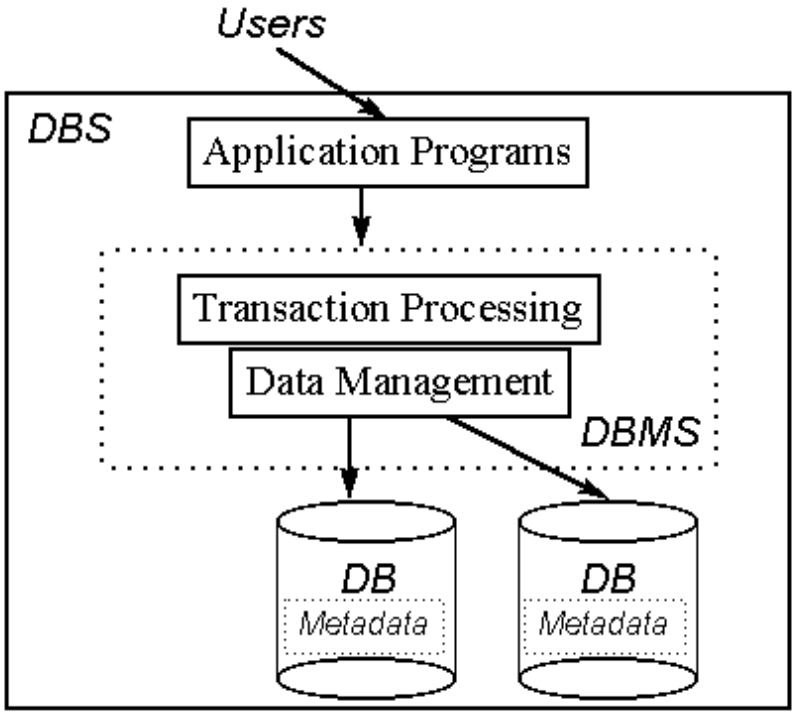


Figure 1.1 Block diagram of a Database system
Source: Baruch College City University of New York

3.2 Advantages of a DBMS

A DBMS can provide:

a. Data Consistency and Integrity - by controlling access and minimizing data duplication
b. Application program independence - by storing data in a uniform fashion
c. Data Sharing - by controlling access to data items, many users can access data concurrently
d. Backup and Recovery
e. Security and Privacy
f. Multiple views of data

3.3 Example Database

Table 1: An Example Database

CustomerID	Name	Address	City	State	AccountNumber	Balance
123	Mr. Sola	12 Lekki	Lagos	LA	0001	4000
123	Mr. Sola	12 Lekki	Lagos	LA	0002	2000
124	Mrs. James	15 Awolowo Ave.	Lagos	LA	0003	1000
125	Mr. Ade	43 Gwagwa Ln.	Maitama	AB	0004	6000
125	Mr. Ade	43 Gwagwa Ln.	Maitama	AB	0005	9000
127	Mr. & Mrs. Bayo	61 Zik Rd.	Garki	AB	0006	500
127	Mr. & Mrs. Bayo	61 Zik Rd.	Garki	AB	0007	800

Activity A

Use table 1 to answer the following questions

1. What happens when a customer moves to a new house?
2. Who should have access to what data in this database?
3. What happens if Mr. and Mrs. Bayo both try and withdraw N500 from account 0006?
4. What happens if the system crashes just as Mr. Ade is depositing his latest paycheck?
5. What data is the customer concerned with?
6. What data is a bank manager concerned with?
7. Send a mailing to all customers with checking accounts having greater than N2000 balance
8. Let all AB customers know of a new branch location

3.4 Brief History of Database Systems

a. **Ancient History:**
   i. Data are not stored on disk; programmer defines both logical data structure and physical structure, such as storage structure, access methods, I/O modes etc.

ii. One data set per program: High data redundancy.
iii. There is no persistence; Random access memory (RAM) is expensive and limited, Programmer productivity low.

b. **1968 File-Based:**
   i. Predecessor of database, data maintained in a flat file.
   ii. Processing characteristics determined by common use of magnetic tape medium.
   iii. Data are stored in files with interface between programs and files.
   iv. Mapping happens between logical files and physical file, one file corresponds to one or several programs.
   v. Various access methods exits, e.g., sequential, indexed, random.
   vi. Requires extensive programming in third-generation language such as COBOL, BASIC.
   vii. **Limitations:**
      1. Separation and isolation: Each program maintains its own set of data, users of one program may not aware of holding or blocking by other programs.
      2. Duplication: Same data is held by different programs, thus, wastes space and resources.
      3. High maintenance costs such as ensuing data consistency and controlling access
      4. Sharing granularity is very coarse
      5. Weak security

c. **1970 presents Era of relational database and Database Management System:** Based on relational calculus, shared collection of logically related data and a description of this data, designed to meet the information needs of an organization; System catalog/metadata provides description of data to enable program-data independence; logically related data comprises entities, attributes, and relationships of an organization's information. Data abstraction allows view level, a way of presenting data to a group of users and logical level, how data is understood to be when writing queries.

d. In 1970, Ted Codd at IBM's San Jose Lab proposed relational models. Two major projects started and both were operational in late 1970s. INGRES at University of California, Berkeley became commercial and followed up POSTGRES which was incorporated into Informix. System R at IBM san Jose Lab, later evolved into DB2, which became one of the first DBMS product based on the relational model. (Oracle produced a similar product just prior to DB2.)
e. 1976: Peter Chen defined the Entity-relationship(ER) model

f. 1980s: Maturation of the relational database technology, more relational based DBMS were developed and SQL standard adopted by ISO and ANSI.

g. 1985: Object-oriented DBMS (OODBMS) develops. Little success commercially because advantages did not justify the cost of converting billions of bytes of data to new format.

h. 1990s: incorporation of object-orientation in relational DBMSs, new application areas, such as data warehousing and OLAP, web and Internet, Interest in text and multimedia, enterprise resource planning (ERP) and management resource planning (MRP).

i. 1991: Microsoft ships access, a personal DBMS created as element of Windows gradually supplanted all other personal DBMS products.

j. 1995: First Internet database applications were introduced.

k. 1997: XML applied to database processing, which solves long-standing database problems. Major vendors begin to integrate XML into DBMS products.

3.5 Contents of a Database

Database consists of:

a. User Data
b. Metadata
c. Indexes
d. Application metadata

3.5.1 User Data

i. Users work with database directly by entering, updating and viewing data.
ii. For our purposes, data will be generally stored in *tables* with some *relationships* between tables.
iii. Each table has one or more *columns*. A set of columns forms a database *record*.
iv. Recall our example database for the bank. What were some problems we discussed?
v. Here is one improvement - split into 2 tables:

Table 2: Customers Table

CustomerID	Name	Address	City	State
123	Mr. Sola	12 Lekki	Lagos	LA
124	Mrs. James	15 Awolowo Ave.	Lagos	LA
125	Mr. Ade	43 Gwagwa Ln.	Maitama	AB

127	Mr. & Mrs. Bayo	61 Zik Rd.	Garki	AB

Table 3: Account Table

CustomerID	AccountNumber	Balance
123	0001	4000
123	0002	2000
124	0003	1000
125	004	6000
125	005	9000
127	006	500
127	007	800

vi. The customers table has 4 records and 5 columns. The Accounts table has 7 records and 3 columns.
vii. Note relationship between the two tables - CustomerID column.
viii. How should we split data into the tables? What are the relationships between the tables?
These are questions that are answered by Database Modeling and Database Design. We shall consider Database modeling in unit 2.

3.5.2 Metadata

Recall that a database is self describing, therefore, Metadata can be described as:

i. Data about data.
ii. Data that describe how user data are stored in terms of table name, column name, data type, length, primary keys, etc.
iii. Metadata are typically stored in *System tables* or *System Catalog* and are typically only directly accessible by the DBMS or by the system administrator.

Have a look at the Database Documentor feature of MS Access (under the tools menu, choose Analyze and then Documentor). This tool queries the system tables to give all kinds of Metadata for tables, etc. in an MS Access database.

3.5.3 Indexes

In keeping with our desire to provide users with several different views of data, indexes provide an alternate means of accessing, sorting and searching data.

An index for our new banking example might include the account numbers in a sorted order.

Indexes allow the database to access a record without having to search through the entire table.

Updating data requires an extra step: The index must also be updated.

Example: Index in a book consists of two things:
1) A Keyword stored in order
2) A *pointer* to the rest of the information. In the case of the book, the pointer is a page number.

3.5.3 Applications Metadata

Many DBMS have storage facilities for forms, reports, queries and other application components.

Applications Metadata is accessed via the database development programs.

Example: Look at the Documentor tool in MS Access. It can also show metadata for Queries, Forms, Reports, etc.

3.6 Data Modeling and Database Design

In this section, we will define the following:

a. **Database Design:** The activity of specifying the *schema* of a database in a given *data model*
b. **Database Schema:** The structure of a database that:

   i. Captures data types, relationships and constraints in data
   ii. Is independent of any application program
   iii. Changes infrequently

c. **Data Model:**
   i. A set of primitives for defining the structure of a database.
   ii. A set of operations for specifying retrieval and updates on a database
   iii. Examples: Relational, Hierarchical, Networked, Object-Oriented

   In this course, we focus on the Relational data model.

d. **Database Instance or State:** The actual data contained in a database at a given time.

3.6.1 The Database Development Process

The following are brief outline describing the database development process.

a. **User needs assessment and requirements gathering:** Determine what the users are looking for, what functions should be supported, how the system should behave.
b. **Data Modeling:** Based on user requirements, form a logical model of the system. This logical model is then converted to a physical data model (tables, columns, relationships, etc.) that will be implemented.
c. **Implementation:** Based on the data model, a database can be created. Applications are then written to perform the required functions.
d. **Testing:** The system is tested using real data.
e. **Deployment:** The system is deployed to users. Maintenance of the system begins.

3.6.2 Designing a Database - A Brief Example

For our Bank example, lets assume that the managers are interested in creating a database to track their customers and accounts.

a. **Tables**
   CUSTOMERS
   CustomerId, Name, Street, City, State, Zip

   ACCOUNTS
   CustomerId, AccountNumber, AccountType, DateOpened, Balance

   Note that we use an *artificial* identifier (a number we make up) for the customer called CustomerId. Given a CustomerId, we can uniquely identify the remaining information. We call CustomerId a **Key** for the CUSTOMERS table.

   - CustomerId is the *key* for the CUSTOMERS table.
   - AccountNumber is the key for the ACCOUNTS table.
   - CustomerId in the ACCOUNTS table is called a *Foreign Key*

b. **Relationships**
   The relationship between CUSTOMERS and ACCOUNTS is by CustomerId. Since a customer may have more than one account at the bank, we call this a *One to Many* relationship. (1:N).

c. **Domains**
   A domain is a set of values that a column may have. Domain also includes the type and length or size of data found in each column.

   CUSTOMERS

Column	Domain

	Data Type	Size
CustomerId (Key)	Integer	20
Name	Character	30
Street	Character	30
City	Character	25
State	Character	2
Zip	Character	2

ACCOUNTS

Column	Domain	
	Data Type	Size
CustomerId (FK)	Integer	20
AccountNumber (Key)	Integer	15
AccountType	Character	2
DateOpened	Date	
Balance	Real	12,2

We use the above information to build a logical model of the database.

This logical model is then converted to a physical model and implemented as tables.

The following is some example data for the Accounts and Customers tables:

Customers Table

CustomerID	Name	Street	City	State	Zip
123	Mr. Sola	12 Lekki	Lagos	LA	01
124	Mrs. James	15 Awolowo Ave.	Lagos	LA	01
125	Mr. Ade	43 Gwagwa Ln.	Maitama	AB	09
127	Mr. & Mrs. Bayo	61 Zik Rd.	Garki	AB	10

Accounts Table

CustomerId	AccountNumber	AccountType	DateOpened	Balance
123	0001	Checking	10/12/08	4000.00

123	0002	Savings	10/12/08	2000.00
124	0003	Savings	01/05/09	1000.00
125	0004	Checking	12/01/09	6000.00
125	0005	Savings	12/01/09	9000.00
127	0006	Savings	08/22/09	500.00
127	0007	Checking	11/13/08	800.00

d. **Business Rules**
Business rules allow us to specify constraints on what data can appear in tables and what operations can be performed on data in tables. For example:

- An account balance can never be negative.
- A Customer can not be deleted if they have an existing (open) account.
- Money can only be transferred from a "Savings" account to a "Checking" account.
- Savings accounts with less than a $500 balance incur a service charge.

Activity B

Briefly explain the following terms:

(a) User data (b) Metadata (c) Indexes (d) Tables (e) Relationship (f) Domains

4.0 Conclusion

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. Database Management System is a software package designed to store and manages databases.

5.0 Summary

In this unit we have learnt that:

i. A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
ii. **Database Management System (DBMS) is** a collection of software programs that are used to define, construct, maintain and manipulate data in a database. A DBMS contains User Data, Metadata, Indexes and Application metadata.
iii. The advantages of DBMS include: data independence and efficient access; reduced application development time; data integrity and security; uniform data administration; concurrent access and recovery from crashes.
iv. A Data Model is a collection of concepts for describing data.

v. The relational model is the most widely used model today
vi. The relational model concept is relation; this is basically a table with rows and columns.
vii. Every relation has a schema which describes the columns or fields.
viii. A database schema is a description of a particular collection of data, using a given model.
ix. Database development process include: User needs assessment and requirements gathering; Data Modeling; Implementation; and Testing.
x. The following three terms are used in database design: Table, Relationship, and Domain.

6.0 Tutor Marked Assignment

1(a) Using Table I, design a database for ABC Bank which would allow them to track their customer and accounts. Pay good attention to tables structures, relationship and domains
(b) Explain the following terms:
i. Key
ii. Foreign key
iii. Domains

2. What are the advantages of using a Database management System?

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 1: Database Management systems Concepts

Unit 2: Data Modeling Overview

1.0 Introduction

This unit is about one of the most critical stages in the development of a computerized information system – the design of data structures and the documentation of that design in a set of data model.

2.0 Objectives

By the end of this unit, you should be able to:

d. Know what data modeling and Entity Relationship is all about
e. Understand the E-R modeling constructs
f. Identify an entity in an E-R relation
g. Know what relationship is in E-R relationship model
h. Draw graph of relations in E-R relationship model
i. Know the advantages of using database management system

3.0 What is Data Modeling?

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern the operations on the objects. To use common analogy, the data model is equivalent to an architect's building plans.

There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. In this unit, we shall focus on the Entity-Relationship approach.

3.1 Data Modeling in the Context of Database Design

Database design is defined as: "designing the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications". The design process roughly follows five steps:

1. planning and analysis
2. conceptual design
3. logical design
4. physical design
5. implementation

3.2 Components of a Data Model

The data model gets its inputs from the planning and analysis stage. Here the modeler, along with system analysts, collects information about the requirements of the database by reviewing the existing documentation and interviewing end-users.

The data model has two outputs. The first is an entity-relationship diagram which represents the data structure in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a

data document. This is a document that describes in detail the data objects, relationships, and rules required by the database.

3.3 Why is data Modeling Important?

The goal of the data model is to make sure that all the data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users.

The data model is also detailed enough to be used by the database developers as a "blueprint" for building the physical database.

The information contained in a data model will be used to define the relational tables, the primary and the foreign keys, stored procedures, and triggers.

A poorly designed database will require more time in the long-run. Without a careful planning you may create a database that omits data required to create critical reports, produces results that are incorrect or inconsistent, and is unable to accommodate changes in user's requirements.

3.4 What Makes a Good Data Model?

The following are the characteristics of a good Data Model:

a. **Completeness**: Does the model support all necessary data?
b. **Non redundancy**: Does the model specify a database in which the same fact could be recorded more than once?
c. **Enforcement of Business Rules**: How accurately does the model reflect and enforce the rules that apply to the business data?
d. **Data Reusability**: Will the data stored in the database be reusable for the purposes beyond those anticipated in the process model?
e. **Stability and Flexibility**: How well will the model cope with possible changes to the business requirements?
f. **Elegance**: Does the data model provide a reasonable neat and simple classification of the data?
g. **Communication**: How effective is the model in supporting communication among the various stakeholders in the design of the system?
h. **Integration**: How will the proposed database fit with the organization's existing and future database?

Activity A

1. What is Data Modeling?

2. Why is Data Modeling important?
3. What are the characteristics of a good data model?

3.5 The Entity-Relationship Model

The Entity-Relationship (ER) model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. Today, ER model is commonly used for database design. For the database designer, the utility of the ER model is:

a. It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.
b. It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.
c. In addition, the model can be used as a design plan by the database developer to implement a data model in specific database management software.

3.6 Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities. E-R Modeling Constructs are: Entity, Relationship, Attributes, and Identifiers

It is important to get used to this terminology and to be able to use it at the appropriate time. For example, in the ER Model, we do not refer to *tables*. Here we call them *entities*.

3.6.1 Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are:

i. EMPLOYEES
ii. PROJECTS
iii. CUSTOMER
iv. ORGANIZATION
v. PART
vi. INGREDIENT
vii. PURCHASE ORDER
viii. CUSTOMER ORDER
PRODUCT
ix. INVOICES

An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An *independent entity* is one that does not rely on another for identification. A *dependent entity* is one that relies on another for identification. The following terms are used with entity:

a. **Entity Occurrence:** An *entity occurrence* (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

   An *instance* of an entity is like a specific example:
   Bill Gates is an Employee of Microsoft
   SPAM is a Product
   Greenpeace is an Organization
   Flour is an ingredient

b. **Associative entities** (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

c. **Subtypes entities** are used in *generalization hierarchies* to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

3.6.2 Attributes

Attributes describe the entity of which they are associated. i.e., properties used to distinguish one entity instance from another.

Attributes of entity EMPLOYEE might include:

i. EmployeeID
ii. First Name
iii. Last Name
iv. Street Address
v. City
vi. Local Government Area
vii. State
viii. Date of First Appointment
ix. Current Status
x. Date of Birth

Attributes of entity PRODUCT might include:

i. ProductID
ii. Product_Description

iii. Weight
iv. Size
v. Cost

A particular instance of an attribute is a **value**. For example, "Chukwudi R. Nnanna" is one value of the attribute Name.

The *domain* of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called *keys*, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

3.6.3 Identifier

Identifier is a special attribute used to identify a specific instance of an entity.

- Typically we look for *unique* identifiers:
- Personal File Number uniquely identifies an EMPLOYEE
- CustomerID uniquely identifies a CUSTOMER
- We can also use two attributes to indicate an identifier: ORDER_NUMBER and LINE_ITEM uniquely identify an item on an order.

3.6.4 Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

employees are assigned to projects

projects have subtasks

departments manage one or more projects

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

(a) Degree of a Relationship*: T*he *degree of a relationship* is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities, are the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

(b) Connectivity and Cardinality

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

i. A *one-to-one* (1:1) relationship is when at most one instance of a entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.
ii. A *one-to-many* (1:N) relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

    a department has many employees

    each employee is assigned to one department

iii. A *many-to-many* (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is:

    employees can be assigned to no more than two projects at the same time;

    projects must have assigned at least three employees

A single employee can be assigned to many projects; conversely, a single project can have assigned to it many employee. Here the cardinality for the relationship between employees and projects is two and the cardinality between project and employee is three. Many-to-many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

(c) Direction

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the *parent entity*; the entity where the relationship terminates is the *child entity*.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

(d) Type

An *identifying relationship* is one in which one of the child entities is also a dependent entity. A *non-identifying relationship* is one in which both entities are independent.

(e) Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either *mandatory* or *optional*. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of mandatory existence is the statement "every project must be managed by a single department". If the instance of the entity is not required, it is optional. An example of optional existence is the statement, "employees may be assigned to work on projects".

3.6.5 Generalization Hierarchies

A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a *supertype* or *generic* entity. The lower-level of entities become the *subtype*, or categories, to the supertype. Subtypes are dependent entities.

Generalization occurs when two or more entities represent categories of the same real-world object. For example, Wages_Employees and Classified_Employees represent categories of the same entity, Employees. In this example, Employees would be the supertype; Wages_Employees and Classified_Employees would be the subtypes.

Subtypes can be either mutually exclusive (disjoint) or overlapping (inclusive). A mutually exclusive category is when an entity instance can be in only one category. The above example is a mutually exclusive category. An employee can either be wages or classified but not both. An overlapping category is when an entity instance may be in two or more subtypes. An example would be a person who works for a university could also be a student at that same university. The completeness constraint requires that all instances of the subtype be represented in the supertype.

Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of nesting is limited only by the constraint of simplicity. Subtype entities may be the parent entity in a relationship but not the child.

3.7 ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. Today, there are a number of notations used; among the more common are Bachman, crow's foot, and IDEFIX.

All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The symbols used for the basic ER constructs are:

i. Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
ii. Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.
iii. Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
iv. Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.
v. Existence is represented by placing a circle or a perpendicular bar on the line. Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

Examples of these symbols are shown in Figure 2.1:

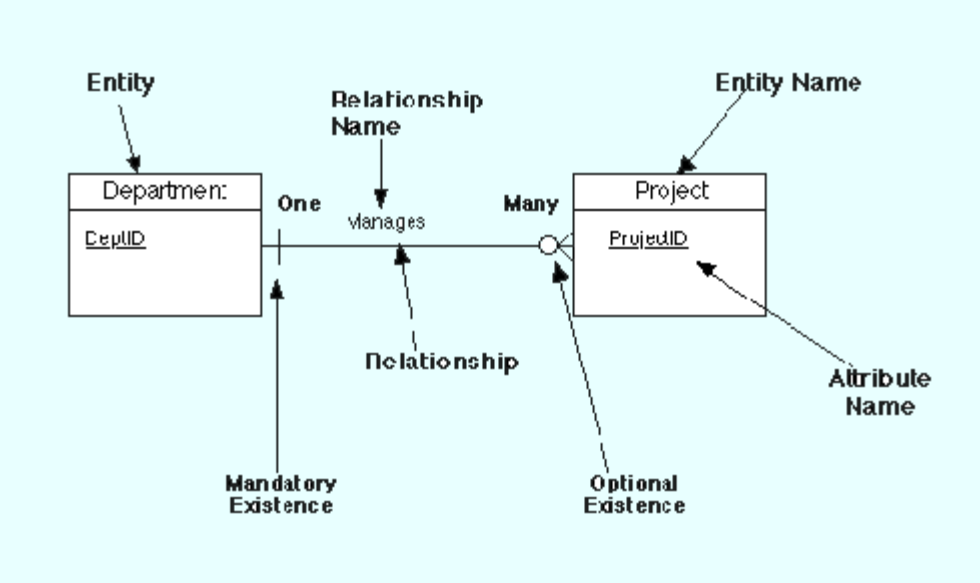


Figure 2.1: ER Notation
Source: http://www.utexas.edu/

Activity B

1. Come up with a list of attributes for each of the entities in section 3.3.1
2. Choose one of your attributes as the identifier for each of the entities.

4.0 Conclusion

The data model is relatively small part of the total systems specification but has a high impact on the quality and useful life of the system. Time spent producing the best possible design is very likely to be repaid many times over in the future.

5.0 Summary

In this unit, we have learnt that:

i. A data model is a plan for building a database. To be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database designer to use to create the physical structure.
ii. The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram.
iii. The basic constructs of the ER model are entities, relationships, and attributes.
iv. Entities are concepts, real or abstract, about which information is collected.

v. Relationships are associations between the entities.
vi. Attributes are properties which describe the entities.

6.0 Tutor Marked Assignment

1. Explain the following E-R Modeling Constructs with examples:
    i. Entity
    ii. Relationship
    iii. Attributes
    iv. Identifiers
2. What do you understand by the term Generalization Hierarchies?

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 1: Database Management systems Concepts

Unit 3: Entity-Relationship Model

1.0 Introduction

The data model is one part of the conceptual design process. The other is the **function model**. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries that will access and perform operations on those tables.

Data modeling is preceded by planning and analysis. The effort devoted to this stage is proportional to the scope of the database. The planning and analysis of a database intended to serve the needs of an enterprise will require more effort than one intended to serve a small workgroup.

The information needed to build a data model is gathered during the requirements analysis. Although not formally considered part of the data modeling stage by some methodologies, in reality the requirements analysis and the ER diagramming part of the data model are done at the same time.

2.0 Objectives

By the end of this unit, you should be able to:

j. Know what data modeling and Entity Relationship is all about
k. Understand the E-R modeling constructs
l. Identify an entity in an E-R relation
m. Know what relationship is in E-R relationship model
n. Draw graph of relations in E-R relationship modelKnow the advantages of using database management system

3.0 Requirements Analysis

The goals of the requirements analysis are:

a. to determine the data requirements of the database in terms of primitive objects
b. to classify and describe the information about these objects
c. to identify and classify the relationships among the objects
d. to determine the types of transactions that will be executed on the database and the interactions between the data and the transactions
e. to identify rules governing the integrity of the data

The modeler, or modelers, works with the end users of an organization to determine the data requirements of the database. Information needed for the requirements analysis can be gathered in several ways:

a. Review of existing documents - such documents include existing forms and reports, written guidelines, job descriptions, personal narratives, and memoranda. Paper documentation is a good way to become familiar with the organization or activity you need to model.
b. Interviews with end users - these can be a combination of individual or group meetings. Try to keep group sessions to under five or six people. If possible, try to have everyone with the same function in one meeting. Use a blackboard, flip charts, or overhead transparencies to record information gathered from the interviews.
c. review of existing automated systems - if the organization already has an automated system, review the system design specifications and documentation

The requirements analysis is usually done at the same time as the data modeling. As information is collected, data objects are identified and classified as either entities, attributes, or relationship; assigned names; and, defined using terms familiar to the end-users. The objects are then modeled and analysed using an ER diagram. The diagram can be reviewed by the modeler and the end-users to determine its completeness and accuracy. If the model is not correct, it is modified, which sometimes requires additional information to be collected. The review and edit cycle continues until the model is certified as correct.

3.1 Steps in Building the Data Model

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies, such as IDEFIX, specify a bottom-up development process where the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. The sequences used in this unit are:

a. Identification of data objects and relationships
b. Drafting the initial ER diagram with entities and relationships
c. Refining the ER diagram
d. Add key attributes to the diagram
e. Adding non-key attributes
f. Diagramming Generalization Hierarchies
g. Validating the model through normalization
h. Adding business and integrity rules to the Model

3.2 Identifying Data Objects and Relationships

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirements analysis for the purpose of:

a. classifying data objects as either entities or attributes

b. identifying and defining relationships between entities
c. naming and defining identified entities, attributes, and relationships
d. documenting this information in the data document

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

While the definitions of the constructs in the ER Model are simple, the model does not address the fundamental issue of how to identify them. Some commonly given guidelines are:

a. entities contain descriptive information
b. attributes either identify or describe entities
c. relationships are associations between entities

3.2.1 Entities

There are various definitions of an entity:

a. An entity is a "thing", "concept" or, object". However, entities can sometimes represent the relationships between two or more objects. This type of entity is known as an associative entity.
b. Entities are objects which contain descriptive information. If an data object you have identified is described by other objects, then it is an entity. If there is no descriptive information associated with the item, it is not an entity. Whether or not a data object is an entity may depend upon the organization or activity being modeled.
c. An entity represents many things which share properties. They are not single things. For example, King Lear and Hamlet are both plays which share common attributes such as name, author, and cast of characters. The entity describing these things would be PLAY, with King Lear and Hamlet being instances of the entity.
d. entities which share common properties are candidates for being converted to generalization hierarchies
e. Entities should not be used to distinguish between time periods. For example, the entities 1st Quarter Profits, 2nd Quarter Profits, etc. should be collapsed into a single entity called Profits. An attribute specifying the time period would be used to categorize by time
f. not every thing the users want to collect information about will be an entity. A complex concept may require more than one entity to represent it. Others "things" users think important may not be entities.

3.2.2 Attributes

Attributes are data objects that either identify or describe entities. Attributes that identify entities are called key attributes. Attributes that describe an entity are called non-key attributes.

Attribute values should be atomic, that is, present a single fact. Having disaggregated data allows simpler programming, greater reusability of data, and easier implementation of changes. Normalization also depends upon the "single fact" rule being followed. Common types of violations include:

a. Simple aggregation - a common example is Person Name which concatenates first name, middle initial, and last name. Another is Address which concatenates, street address, city, and zip code. When dealing with such attributes, you need to find out if there are good reasons for decomposing them. For example, do the end-users want to use the person's first name in a form letter? Do they want to sort by zip code?
b. Complex codes - these are attributes whose values are codes composed of concatenated pieces of information. An example is the code attached to automobiles and trucks. The code represents over 10 different pieces of information about the vehicle. Unless part of an industry standard, these codes have no meaning to the end user. They are very difficult to process and update.
c. Text blocks - these are free-form text fields. While they have a legitimate use, an over reliance on them may indicate that some data requirements are not met by the model.
d. mixed domains - this is where a value of an attribute can have different meaning under different conditions

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favor are:

a. derived data is often important to both managers and users and therefore should be included in the data model
b. it is just as important, perhaps more so, to document derived attributes just as you would other attributes
c. including derived attributes in the data model does not imply how they will be implemented

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favor argue that many organizations have a long history of using coded attributes, that codes save space, and improve flexibility in that values can be easily added or modified by means of look-up tables.

3.2.3 Relationships

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities. For example:

> Employees **are assigned** to projects

As relationships are identified they should be classified in terms of cardinality, optionality, direction, and dependence. As a result of defining the relationships, some relationships may be dropped and new relationships added. Cardinality quantifies the relationships between entities by measuring how many instances of one entity are related to a single instance of another. To determine the cardinality, assume the existence of an instance of one of the entities. Then determine how many specific instances of the second entity could be related to the first. Repeat this analysis reversing the entities. For example:

> employees **may be assigned** to no more than three projects at a time;
> every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example:

> An employee **may** be assigned to a project

Mandatory relationships, on the other hand, are indicated by words such as must have. For example:

> a student **must** register for at least three course each semester

In the case of the specific relationship form (1:1 and 1:M), there is always a parent entity and a child entity. In one-to-many relationships, the parent is always the entity with the cardinality of one. In one-to-one relationships, the choice of the parent entity must be

made in the context of the business being modeled. If a decision cannot be made, the choice is arbitrary.

3.2.4 Naming Data Objects

The names should have the following properties:

- unique
- have meaning to the end-user
- contain the minimum number of words needed to uniquely and accurately describe the object

For entities and attributes, names are singular nouns while relationship names are typically verbs.

Some authors advise against using abbreviations or acronyms because they might lead to confusion about what they mean. Other believes using abbreviations or acronyms are acceptable provided that they are universally used and understood within the organization.

You should also take care to identify and resolve synonyms for entities and attributes. This can happen in large projects where different departments use different terms for the same thing.

3.3 Developing the Basic Schema

Once entities and relationships have been identified and defined, the first draft of the entity relationship diagram can be created. This section introduces the ER diagram by demonstrating how to diagram binary relationships. Recursive relationships are also shown.

3.3.1 Binary Relationships

Figure 3.1 shows examples of how to diagram one-to-one, one-to-many, and many-to-many relationships.

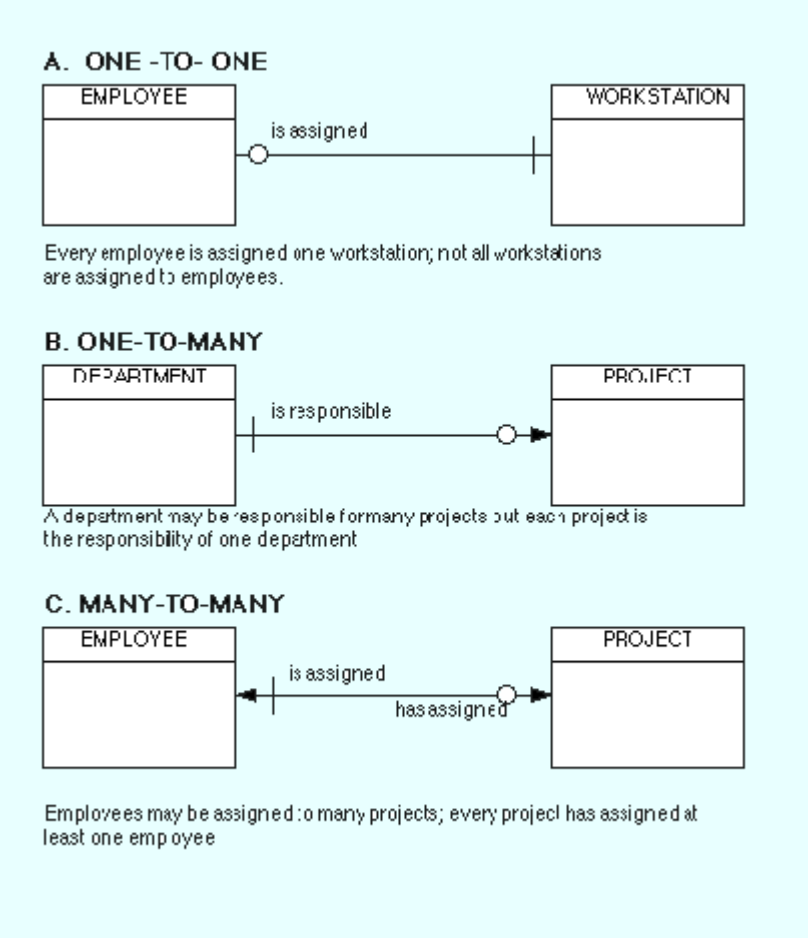


Figure 3.1: Example of Binary Relationships
Source: http://www.utexas.edu/its/

(a) One-To-One: Figure 3.1A shows an example of a one-to-one diagram. Reading the diagram from left to right represents the relationship every employee is assigned a workstation. Because every employee must have a workstation, the symbol for mandatory existence—in this case the crossbar—is placed next to the WORKSTATION entity. Reading from right to left, the diagram shows that not all workstation are assigned to employees. This condition may reflect that some workstations are kept for spares or for loans. Therefore, we use the symbol for optional existence, the circle, next to EMPLOYEE. The cardinality and existence of a relationship must be derived from the "business rules" of the organization. For example, if all workstations owned by an organization were assigned to employees, then the circle would be replaced by a crossbar to indicate mandatory existence. One-to-one relationships are rarely seen in "real-world" data models.

(b) One-To-Many: Figure 3.1B shows an example of a one-to-many relationship between DEPARTMENT and PROJECT. In this diagram, DEPARTMENT is considered the parent entity while PROJECT is the child. Reading from left to right, the diagram represents departments may be responsible for many projects. The optionality of the

relationship reflects the "business rule" that not all departments in the organization will be responsible for managing projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

(c) Many-To-Many: Figure 1C shows a many-to-many relationship between EMPLOYEE and PROJECT. An employee **may** be assigned to many projects; each project **must** have many employee Note that the association between EMPLOYEE and PROJECT is optional because, at a given time, an employee may not be assigned to a project. However, the relationship between PROJECT and EMPLOYEE is mandatory because a project must have at least two employees assigned. Many-To-Many relationships can be used in the initial drafting of the model but eventually must be transformed into two one-to-many relationships. The transformation is required because many-to-many relationships cannot be represented by the relational model. The process for resolving many-to-many relationships is discussed in the next section.

3.3.2 Recursive relationships

A recursive relationship is an entity that is associated with itself. Figure 3.2 shows an example of the recursive relationship.

An employee may manage many employees and each employee is managed by one employee.

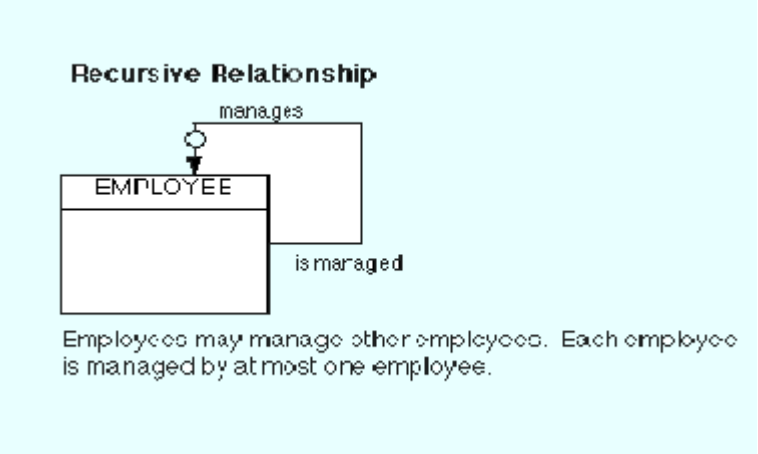


Figure 3.2: Example of Recursive Relationship
Source: http://www.utexas.edu/its/

3.4 Refining the Entity-Relationship Diagram

This section discusses three basic rules for modeling relationships

3.4.1 Entities Must Participate In Relationships

Entities cannot be modeled unrelated to any other entity. Otherwise, when the model was transformed to the relational model, there would be no way to navigate to that table. The exception to this rule is a database with a single table.

3.4.2 Resolve Many-To-Many Relationships

Many-to-many relationships cannot be used in the data model because they cannot be represented by the relational model. Therefore, many-to-many relationships must be resolved early in the modeling process. The strategy for resolving many-to-many relationship is to replace the relationship with an association entity and then relate the two original entities to the association entity. This strategy is demonstrated below Figure 3.2A shows the many-to-many relationship:

Employees may be assigned to many projects.
Each project must have assigned to it more than one employee.

In addition to the implementation problem, this relationship presents other problems. Suppose we wanted to record information about employee assignments such as who assigned them, the start date of the assignment, and the finish date for the assignment. Given the present relationship, these attributes could not be represented in either EMPLOYEE or PROJECT without repeating information. The first step is to convert the relationship **assigned to** to a new entity we will call ASSIGNMENT. Then the original entities, EMPLOYEE and PROJECT, are related to this new entity preserving the cardinality and optionality of the original relationships. The solution is shown in Figure 3.2B.

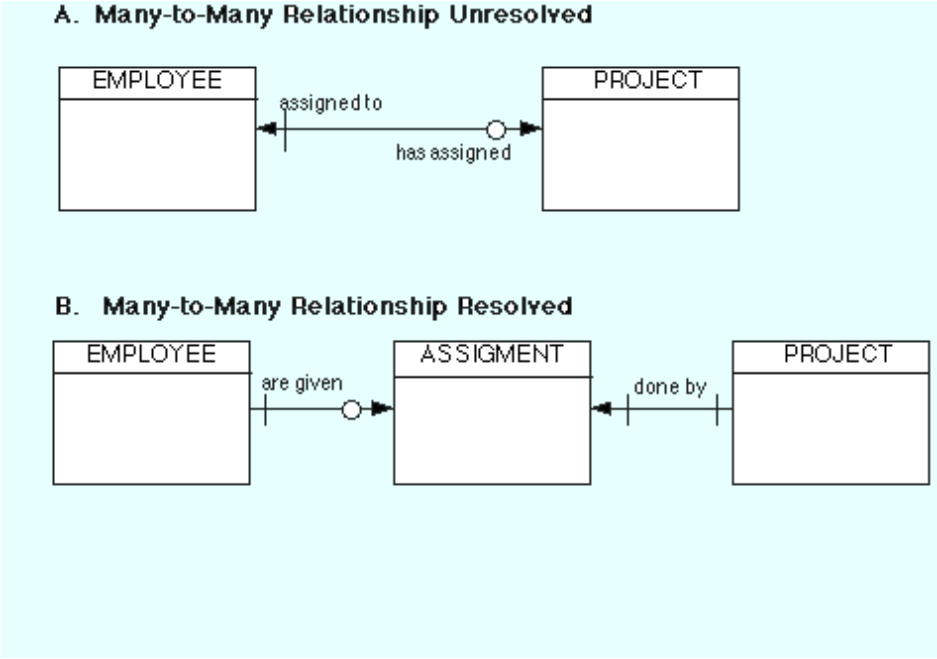


Figure 3.2: Resolution of a Many-To-Many Relationship
Source: http://www.utexas.edu/its/

Notice that the schema changes the semantics of the original relation to

employees **may be given** assignments to projects
and projects must **be done by** more than one employee assignment.

3.4.3 Eliminate redundant relationships

A redundant relationship is a relationship between two entities that is equivalent in meaning to another relationship between those same two entities that may pass through an intermediate entity. For example, Figure 3.3A shows a redundant relationship between DEPARTMENT and WORKSTATION. This relationship provides the same information as the relationships DEPARTMENT has EMPLOYEES and EMPLOYEEs assigned WORKSTATION. Figure 3.3B shows the solution which is to remove the redundant relationship DEPARTMENT assigned WORKSTATIONS.

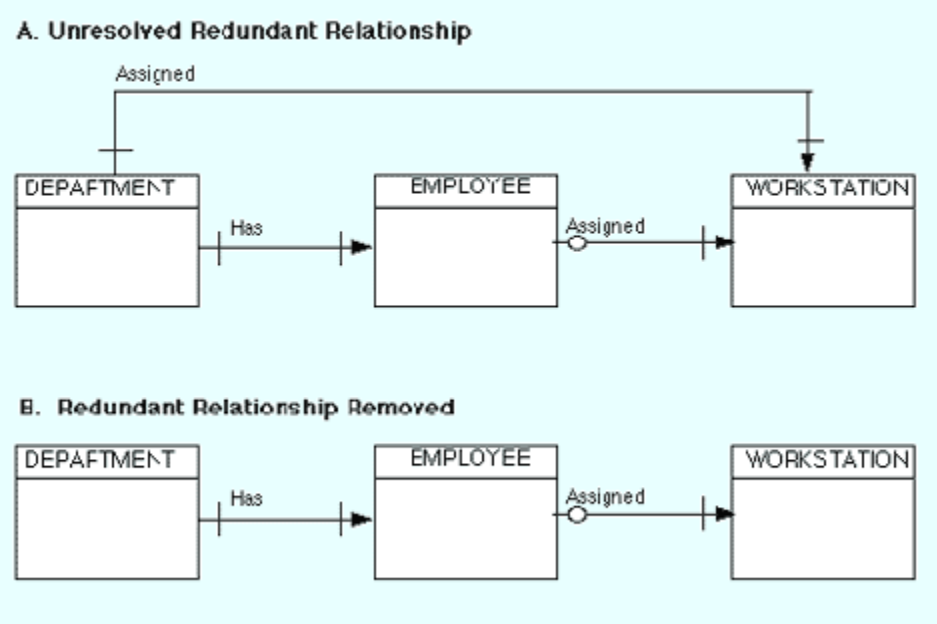


Figure 3.3: Removing A Redundant Relationship
Source: http://www.utexas.edu/its/

3.5 Primary and Foreign Keys

Primary and foreign keys are the most basic components on which relational theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities. The next step in building the basic data model to

1. identify and define the primary key attributes for each entity
2. validate primary keys and relationships
3. migrate the primary keys to establish foreign keys

3.5.1 Define Primary Key Attributes

The primary key is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.

To qualify as a primary key for an entity, an attribute must have the following properties:

- it must have a non-null value for each instance of the entity
- the value must be unique for each instance of an entity
- the values must not change or become null during the life of each entity instance

In this section, we shall discuss the following:

(a) Candidate Key

In some instances, an entity will have more than one attribute that can serve as a primary key. Any key or minimum set of keys that could be a primary key is called a candidate key. Once candidate keys are identified, choose one, and only one, primary key for each entity. Choose the identifier most commonly used by the user as long as it conforms to the properties listed above. Candidate keys which are not chosen as the primary key are known as alternate keys.

An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name.

Name is the least desirable candidate. While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. Moreover, there is the possibility that an employee's name could change because of marriage. Employee ID would be a good candidate as long as each employee was assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

(b) Composite Keys

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that made up of more than one attribute is known as a composite key. Figure 3.4 shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ID.

WORK

Employee ID	Project ID	Hours_Worked
01	01	200
01	02	120
02	01	50
02	03	120
03	03	100
03	04	200

Figure 3.4: Example of Composite Key
Source: http://www.utexas.edu/its/

(c) Artificial Keys

An artificial keyis one that has no meaning to the business or organization. Artificial keys are permitted when
1) no attribute has all the primary key properties, or
2) the primary key is large and complex.

(d) Primary Key Migration

Dependent entities, entities that depend on the existence of another entity for their identification, inherit the entire primary key from the parent entity. Every entity within a generalization hierarchy inherits the primary key of the root generic entity.

(e) Define Key Attributes

Once the keys have been identified for the model, it is time to name and define the attributes that have been used as keys.

There is no standard method for representing primary keys in ER diagrams. For this document, the name of the primary key followed by the notation (PK) is written inside the entity box. An example is shown in Figure 3.5A.

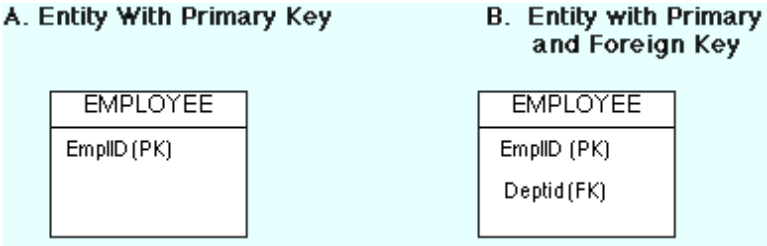


Figure 3.5: Entities with Key Attributes
Source: http://www.utexas.edu/its/

(f) Validate Keys and Relationships

Basic rules governing the identification and migration of primary keys are:

a. Every entity in the data model shall have a primary key whose values uniquely identify entity instances.
b. The primary key attribute cannot be optional (i.e., have null values).
c. The primary key cannot have repeating values. That is, the attribute may not have more than one value at a time for a given entity instance is prohibited. This is known as the No Repeat Rule.
d. Entities with compound primary keys cannot be split into multiple entities with simpler primary keys. This is called the Smallest Key Rule.
e. Two entities may not have identical primary keys with the exception of entities within generalization hierarchies.
f. The entire primary key must migrate from parent entities to child entities and from supertype, generic entities, to subtypes, category entities.

3.5.2 Foreign Keys

A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity. Every relationship in the model must be supported by a foreign key. In this section we shall discuss the following:

(a) Identifying Foreign Keys

Every dependent and category (subtype) entity in the model must have a foreign key for each relationship in which it participates. Foreign keys are formed in dependent and subtype entities by migrating the entire primary key from the parent or generic entity. If the primary key is composite, it may not be split.

(b) Foreign Key Ownership

Foreign key attributes are not considered to be owned by the entities to which they migrate, because they are reflections of attributes in the parent entities. Thus, each attribute in an entity is either owned by that entity or belongs to a foreign key in that entity.

If the primary key of a child entity contains all the attributes in a foreign key, the child entity is said to be "identifier dependent" on the parent entity, and the relationship is called an "identifying relationship." If any attributes in a foreign key do not belong to the child's primary key, the child is not identifier dependent on the parent, and the relationship is called "non identifying."

(c) Diagramming Foreign Keys

Foreign keys attributes are indicated by the notation (FK) beside them. An example is shown in Figure 3.5 (B) above.

Activity A

1. Explain the following terms in relation to data objects
   i. Primary Key
   ii. Candidate Key
   iii. Composite Key
   iv. Artificial Key
   v. Foreign Key
2. What do you understand by Binary relationship in database design?

3.6 Adding Attributes to the Model

Non-key attributes describe the entities to which they belong. In this section, we discuss the rules for assigning non-key attributes to entities and how to handle multivalued attributes.

3.6.1 Relate attributes to entities

Non-key attributes can be in only one entity. Unlike key attributes, non-key attributes never migrate, and exist in only one entity, from parent to child entities.

The process of relating attributes to the entities begins by the modeler, with the assistance of the end-users, placing attributes with the entities that they appear to describe. Once this is completed, the assignments are validated by the formal method of normalization.

Before beginning formal normalization, the rule is to place non-key attributes in entities where the value of the primary key determines the values of the attributes. In general, entities with the same primary key should be combined into one entity. Some other guidelines for relating attributes to entities are given below.

3.6.2 Parent-Child Relationships

- With parent-child relationships, place attributes in the parent entity where it makes sense to do so (as long as the attribute is dependent upon the primary key)
- If a parent entity has no non-key attributes, combine the parent and child entities.

3.6.3 Multivalued Attributes

If an attribute is dependent upon the primary key but is multivalued, has more than one value for a particular value of the key, reclassify the attribute as a new child entity. If the multivalued attribute is unique within the new entity, it becomes the primary key. If not migrate the primary key from the original, now parent, entity.

For example, assume an entity called PROJECT with the attributes Proj_ID (the key), Proj_Name, Task_ID, Task_Name

PROJECT

Proj_ID	Proj_Name	Task_ID	Task_Name
01	A	01	Analysis
01	A	02	Design
01	A	03	Programming
01	A	04	Tuning
02	B	01	Analysis

Figure 3.6: Example of multivalued Attributes

Source: http://www.utexas.edu/its/

Task_ID and Task_Name have multiple values for the key attribute. The solution is to create a new entity, let's call it TASK and make it a child of PROJECT. Move Task_ID and Task_Name from PROJECT to TASK. Since neither attribute uniquely identifies a task, the final step would be to migrate Proj_ID to TASK.

3.6.4 Attributes That Describe Relations

In some cases, it appears that an attribute describes a relationship rather than an entity (in the Chen notation of ER diagrams this is permissible). For example,

A MEMBER **borrows** BOOKS.

Possible attributes are the date the books were checked out and when they are due. Typically, such a situation will occur with a many-to-many relationship and the solution is the same. Reclassify the relationship as a new entity which is a child to both original entities. In some methodologies, the newly created is called an associative entity.

3.6.5 Derived Attributes and Code Values

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favor are:

- derived data is often important to both managers and users and therefore should be included in the data model.
- it is just as important, perhaps more so, to document derived attributes just as you would other attributes
- including derived attributes in the data model does not imply how they will be implemented.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favor argue that many organizations have a long history of using coded attributes, that codes save space, and improve flexibility in that values can be easily added or modified by means of look-up tables.

3.7 Generalization Hierarchies

Up to this point, we have discussed describing an object, the entity, by its shared characteristics, the attributes. For example, we can characterize an employee by their employee id, name, job title, and skill set.

Another method of characterizing entities is by both similarities and differences. For example, suppose an organization categorizes the work it does into internal and external projects. Internal projects are done on behalf of some unit within the organization. External projects are done for entities outside of the organization. We can recognize that both types of projects are similar in that each involves work done by employees of the organization within a given schedule. Yet we also recognize that there are differences between them. External projects have unique attributes, such as a customer identifier and the fee charged to the customer. This process of categorizing entities by their similarities and differences is known as generalization.

3.7.1 Description

A generalization hierarchy is a structured grouping of entities that share common attributes. It is a powerful and widely used method for representing common characteristics among entities while preserving their differences. It is the relationship between an entity and one or more refined versions. The entity being refined is called the supertype and each refined version is called the subtype. The general form for a generalization hierarchy is shown in Figure 3.7

Generalization hierarchies should be used when:

- A large number of entities appear to be of the same type,
- Attributes are repeated for multiple entities, or
- The model is continually evolving. Generalization hierarchies improve the stability of the model by allowing changes to be made only to those entities germane to the change and simplify the model by reducing the number of entities in the model.

3.7.2 Creating a Generalization Hierarchy

To construct a generalization hierarchy, all common attributes are assigned to the supertype. The supertype is also assigned an attribute, called a discriminator, whose values identify the categories of the subtypes. Attributes unique to a category, are assigned to the appropriate subtype. Each subtype also inherits the primary key of the supertype. Subtypes that have only a primary key should be eliminated. Subtypes are related to the supertypes through a one-to-one relationship.

3.7.3 Types of Hierarchies

A generalization hierarchy can either be overlapping or disjoint. In an overlapping hierarchy an entity instance can be part of multiple subtypes. For example, to represent people at a university you have identified the supertype entity PERSON which has three subtypes, FACULTY, STAFF, and STUDENT. It is quite possible for an individual to be in more than one subtype, a staff member who is also registered as a student, for example.

In a disjoint hierarchy, an entity instance can be in only one subtype. For example, the entity EMPLOYEE, may have two subtypes, CLASSIFIED and WAGES. An employee may be one type or the other but not both. Figure 1 shows A) overlapping and B) disjoint generalization hierarchy.

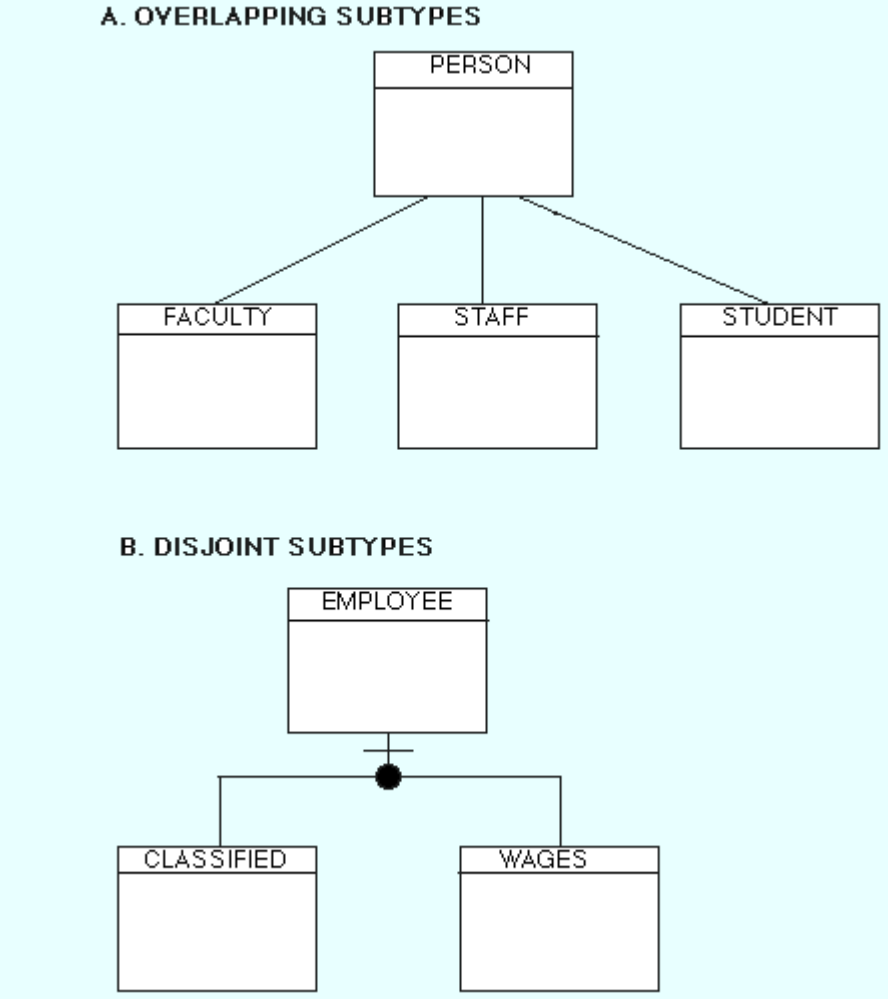


Figure 3.7: Examples of Generalization Hierarchies
Source: http://www.utexas.edu/its/

3.7.4 Rules

The primary rule of generalization hierarchies is that each instance of the supertype entity must appear in at least one subtype; likewise, an instance of the subtype must appear in the supertype.

Subtypes can be a part of only one generalization hierarchy. That is, a subtype can not be related to more than one supertype. However, generalization hierarchies may be nested by having the subtype of one hierarchy be the supertype for another.

Subtypes may be the parent entity in a relationship but not the child. If this were allowed, the subtype would inherit two primary keys.

3.8 Add Data Integrity Rules

Data integrity is one of the cornerstones of the relational model. Simply stated data integrity means that the data values in the database are correct and consistent.

Data integrity is enforced in the relational model by **entity** and **referential** integrity rules. Although not part of the relational model, most database software enforces attribute integrity through the use of domain information.

3.8.1 Entity Integrity

The entity integrity rule states that for every instance of an entity, the value of the primary key must exist, be unique, and cannot be null. Without entity integrity, the primary key could not fulfill its role of uniquely identifying each instance of an entity.

3.8.2 Referential Integrity

The referential integrity rule states that every foreign key value must match a primary key value in an associated table. Referential integrity ensures that we can correctly navigate between related entities.

3.8.3 Insert and Delete Rules

A foreign key creates a hierarchical relationship between two associated entities. The entity containing the foreign key is the child, or dependent, and the table containing the primary key from which the foreign key values are obtained is the parent.

In order to maintain referential integrity between the parent and child as data is inserted or deleted from the database certain insert and delete rules must be considered.

(a) Insert Rules

Insert rules commonly implemented are:

i. **Dependent**. The dependent insert rule permits insertion of child entity instance only if matching parent entity already exists.
ii. **Automatic**. The automatic insert rule always permits insertion of child entity instance. If matching parent entity instance does not exist, it is created.
iii. **Nullify**. The nullify insert rule always permits the insertion of child entity instance. If a matching parent entity instance does not exist, the foreign key in child is set to null.
iv. **Default**. The default insert rule always permits insertion of child entity instance. If a matching parent entity instance does not exist, the foreign key in the child is set to previously defined value.

v. **Customized.** The customized insert rule permits the insertion of child entity instance only if certain customized validity constraints are met.
vi. **No Effect.** This rule states that the insertion of child entity instance is always permitted. No matching parent entity instance need exist, and thus no validity checking is done.

(b) Delete Rules

i. **Restrict**. The restrict delete rule permits deletion of parent entity instance only if there are no matching child entity instances.
ii. **Cascade.** The cascade delete rule always permits deletion of a parent entity instance and deletes all matching instances in the child entity.
iii. **Nullify**. The nullify delete rules always permits deletion of a parent entity instance. If any matching child entity instances exist, the values of the foreign keys in those instances are set to null.
iv. **Default.** The default rule always permits deletion of a parent entity instance. If any matching child entity instances exist, the value of the foreign keys are set to a predefined default value.
v. **Customized.** The customized delete rule permits deletion of a parent entity instance only if certain validity constraints are met.
vi. **No Effect.** The no effect delete rule always permits deletion of a parent entity instance. No validity checking is done.

3.9 Domains

A domain is a valid set of values for an attribute which enforce that values from an insert or update make sense. Each attribute in the model should be assigned domain information which includes:

a. **Data Type—**Basic data types are integer, decimal, or character. Most data bases support variants of these plus special data types for date and time.
b. **Length—**This is the number of digits or characters in the value. For example, a value of 5 digits or 40 characters.
c. **Date Format—**The format for date values such as dd/mm/yy or yy/mm/dd
d. **Range—**The range specifies the lower and upper boundaries of the values the attribute may legally have
e. **Constraints—**Are special restrictions on allowable values. For example, the Beginning_Pay_Date for a new employee must always be the first work day of the month of hire.
f. **Null support—**Indicates whether the attribute can have null values

g. **Default value (if any)—**The value an attribute instance will have if a value is not entered.

3.9.1 Primary Key Domains

The values of primary keys must be unique and nulls are not allowed.

3.9.2 Foreign Key Domains

The data type, length, and format of primary keys must be the same as the corresponding primary key. The uniqueness property must be consistent with relationship type. A one-to-one relationship implies a unique foreign key; a one-to-many relationship implies a non-unique foreign key.

Activity B

1. What do you understand by the term Generalization Hierarchy?
2. Explain the following terms:
   i. Entity Integrity
   ii. Referential integrity
   iii. Entity
   iv. Attributes
   v. Relationship
   vi. Supertypes

4.0 Conclusion

Data modeling stage is a very important stage in database or information system design. There would be problem either now or in the nearest future if proper and complete data modeling is not done.

5.0 Summary

In this unit, we have learnt that:

i. Data modeling must be preceded by planning and analysis. Planning defines the goals of the database, explains why the goals are important, and sets out the path by which the goals will be reached. Analysis involves determining the requirements of the database. This is typically done by examining existing documentation and interviewing users.
ii. An effective data model completely and accurately represents the data requirements of the end users. It is simple enough to be understood by the end user yet detailed enough to be used by a database designer to build the database. The model eliminates redundant data, it is independent of any hardware and

software constraints, and can be adapted to changing requirements with a minimum of effort.

iii. Data modeling is a bottom up process. A basic model, representing entities and relationships, is developed first. Then detail is added to the model by including information about attributes and business rules.

iv. The first step in creating the data model is to analyze the information gathered during the requirements analysis with the goal of identifying and classifying data objects and relationships.

v. The Entity-Relationship diagram provides a pictorial representation of the major data objects, the entities, and the relationships between them.

vi. Generalization hierarchies are a structure that enables the modeler to represent entities that share common characteristics but also have differences

6.0 Tutor Marked Assignment

1. Draw an entity-relationship model diagram that captures the student grading system requirements stated below. Once done, **validate the model against the requirements** to make sure nothing was missed.

i. Each student has a first and last name, and a student number.

ii. Each course has a course number (e.g. CIT843) and a title.

iii. A course will have multiple offerings, identified by year, term, and possibly section.

iv. A student may enroll in multiple course offerings.

v. Each course offering divides its overall evaluation into one or more components (e.g. Assignments, Quizzes, Seminars, Final Exam), each weighted some specified fraction of the offering's final grade (e.g. assignments are worth 35%, quizzes worth 35% and the final exam worth 30%).

vi. Each component is made up of one or more graded items (e.g. assignment #3 is a single graded item).

vii. Each graded item records the order number of the item within its component, the date of evaluation, and the maximum mark possible (e.g. quiz number four will be held October 19, 2006, and it is out of a total of 10 marks).

viii. A student's mark is recorded for each graded item in a course offering.

ix. When evaluating some components, one or more of the graded items with the lowest marks are dropped from the calculation.

x. In extenuating circumstances, an instructor may drop a student's mark from his or her evaluation.

xi. At the end of the term, the student's final mark in a course offering is converted to a letter grade and GPA.

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 1: Database Management systems Concepts

Unit 4: Relational Database Integrity: Conversion from E-R Model to Relational Model

1.0 Introduction

Previously, we covered modeling the user's view as an E-R diagram, and Entities, Relationships, Attributes and Identifiers were used.

We now need to convert this conceptual representation to a model that can be implemented directly in a database.

The relational model was formally introduced by Dr. E. F. Codd in 1970 and has evolved since then, through a series of writings. The model provides a simple, yet rigorously defined, concept of how users perceive data. The relational model represents data in the form of two-dimension tables. Each table represents some real-world person, place, thing, or event about which information is collected. A relational database is a collection of two-dimensional tables. The organization of data into relational tables is known as the **logical view** of the database. That is, the form in which a relational database presents data to the user and the programmer. The way the database software physically stores the data on a computer disk system is called the **internal view**.

2.0 Objectives

This unit discusses the basic concepts—data structures, relationships, and data integrity—that are the basis of the relational model.

By the end of this unit, you should be able to:

a. Explain Data Structure and Terminology
b. Know Relational Model Notation
c. Know some of the properties of Relational Tables
d. Explain Relationships and Keys
e. Understand Data Integrity with respect to Relational Model

3.0 Data Structure and Terminology

In the relational model, a database is a collection of relational tables. A relational table is a flat file composed of a set of named columns and an arbitrary number of unnamed rows. The columns of the tables contain information about the table. The rows of the table represent occurrences of the "thing" represented by the table. A data value is stored in the intersection of a row and column. Each named column has a domain, which is the set of values that may appear in that column. **Figure 4.1** shows the relational tables for a simple bibliographic database that stores information about book title, authors, and publishers.

A Relational Data Base

AUTHOR

au_id	au_lname	au_fname	address	city	state
172-32-1176	White	Johnson	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	22 Cleveland Av. #14	San Jose	CA
274-80-9391	Straight	Dean	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	6223 Bateman St.	Berkeley	CA
427-17-2319	Dull	Ann	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	18 Broadway Av.	San Francisco	CA

TITLE

title_id	title	type	price	pub_id
BU1032	The Busy Executive's Database Guide	business	19.99	1389
BU1111	Cooking with Computers	business	11.95	1389
BU2075	You Can Combat Computer Stress!	business	2.99	736
BU7832	Straight Talk About Computers	business	19.99	1389
MC2222	Silicon Valley Gastronomic Treats	mod_cook	19.99	877
MC3021	The Gourmet Microwave	mod_cook	2.99	877
MC3026	The Psychology of Computer Cooking	UNDECIDED		877
PC1035	But Is It User Friendly?	popular_comp	22.95	1389
PC8888	Secrets of Silicon Valley	popular_comp	20	1389
PC9999	Net Etiquette	popular_comp		1389
PS2091	Is Anger the Enemy?	psychology	10.95	736

PUBLISHER

pub_id	pub_name	city
736	New Moon Books	Boston
877	Binnet & Hardley	Washington
1389	Algodata Infosystems	Berkeley
1622	Five Lakes Publishing	Chicago
1756	Ramona Publishers	Dallas
9901	GGG&G	München
9952	Scootney Books	New York
9999	Lucerne Publishing	Paris

AUTHOR_TITLE

au_id	title_id
172-32-1176	PS3333
213-46-8915	BU1032
213-46-8915	BU2075
238-95-7766	PC1035
267-41-2394	BU1111
267-41-2394	TC7777
274-80-9391	BU7832
409-56-7008	BU1032
427-17-2319	PC8888
472-27-2349	TC7777

Figure 4.1: Relational Tables
Source: http://www.utexas.edu/

There are alternate names used to describe relational tables. Some manuals use the terms tables, fields, and records to describe relational tables, columns, and rows, respectively. The formal literature tends to use the mathematical terms, relations, attributes, and tuples. Figure 4.2 summarizes these naming conventions.

ER Model	Relational Model	Database	Traditional Programmer
Entity	Relation	Table	File
Entity Instance	Tuple	Row	Record
Attribute	Attribute	Column	Field
Identifier	Key	Key	Key (or link)

Figure 4.2: Terminology
Source: http://cisnet.baruch.cuny.edu/

3.1 Schema Conversion Rules

Schema conversion is the process of translating an entity-relationship model to a relational schema. A relational **schema** defines the structure of a relational database, not the data that it will contain. Unlike an E-R model, a schema can be directly encoded into an RDBMS using something called the SQL Data Definition Language (DDL).

We will always follow this procedure from start to finish when converting a model to a schema.

i. Convert all entities to tables. The entity name becomes the table name, and the entity attributes become the table columns.
ii. Find the candidate keys for each table, and from them choose a primary key for each table (if possible).
iii. Replace one-to-one, one-to-many, and subtype entity relationships with foreign key columns in the appropriate tables.
iv. Replace many-to-many entity relationships with a new join table that contains foreign key columns of the related tables.
v. Based on the participation of the entity relationships, set the column datatype for the foreign keys to allow or disallow NULL values.
vi. Now that the foreign key columns are in place, find the candidate keys for the tables again (including any newly-added tables), and select a primary key from these. Add surrogate keys if necessary.
vii. Write down the functional dependencies between columns for each table and validate the tables against the COMP210 interpretation of the first 5 normal forms for potential modification anomalies. If there are problems, revisit the E-R model, make corrections, and begin the schema conversion procedure again from the top.
viii. Translate the validated schema into SQL DDL and create the tables, indices, and any referential or unique integrity constraints, in an RDBMS.

3.2 NULL Values

Relational databases introduce the useful, but occasionally misused, concept of a **NULL** value. NULL should be thought of the absence of any meaningful value: it is not the same as zero, or false, or an empty string. Bear in mind that any mathematical operation including a NULL will always result in NULL: 2 + 0 = 2, but 2 + NULL is just NULL again. Note that the equality operator (=) can't be used with NULL, because NULL does not equal NULL, and even more strangely, NULL does not equal NULL.

NULL values are most often used in columns to indicate either Unknown or Not Applicable. Of the two, Not Applicable is the more "proper" interpretation, but both uses are common.

A column in a table may be declared to either allow or disallow NULL values. Despite their utility, columns that allow NULL values should be kept to an absolute minimum (just as we tried to minimize the number of E-R relationships with optional participation).

An optional attribute in the E-R model will convert to a column that accepts NULL values (that's why we tried to minimize the number of those too).

3.3 Keys

Keys are very important to relational schemas; much more so than identifiers to E-R models.

3.3.1 Candidate Keys

A **candidate key** is the set of one or more columns in a table that uniquely identifies a row in that table. A table may have many candidate keys, and by definition will always have one (relational theory requires that all rows in a table be distinct).

3.3.2 Primary Keys

A single **primary key** is chosen from a table's candidate keys. Every table must have one, and only one, primary key. Primary keys have two important restrictions:

- A primary key must not contain any columns that allow NULL values.
- The value of a primary key, whether composed of one or many columns, must never change.

As long as these restrictions are followed, the choice of primary key is largely a design decision, often influenced by either performance or business concerns.

3.3.3 Foreign Keys

A **foreign key** is the set of one or more columns in a table that together are the designated primary key of another, related table. Relational schemas, oddly enough, have no concept of a relationship as in the E-R model. Instead, foreign keys are used to define the relationships between tables. A table may have many foreign keys, and the column or columns making up the foreign key may be included in the table's primary key. A column that is part of a foreign key does not have to use the same name as that used in the primary key of the related table, but it usually does.

3.3.4 Surrogate Keys

A **surrogate key** (a.k.a. artificial key) is a system-supplied unique value for a column that will serve as a table's primary key. The value of a surrogate key has no business meaning and is intended only as a means for establishing relationships between tables (through foreign keys).

There are three main reasons for using surrogate keys:

i. Performance: If the primary key for a table is a long text string, the RDBMS will have to create an enormous index for the table. Foreign keys that reference this table will be similarly huge. Surrogate keys are usually 32-bit integers and can therefore be manipulated very efficiently.
ii. Primary keys must not change values: if, for example, the primary key for a table is an employee's full name, the values of the primary key could potentially change (through marriage, or for celebrity-induced reasons: "The Artist Formerly Known As Prince"). A surrogate key is not affected by such changes, *particularly if the users of the system never see the surrogate key values.*
iii. No other primary key can be found for a table.

3.4 Schema Diagram Notation

The notation used for relational schemas is arbitrary at best. There are no standards. I have used a simplified version of Crow's Foot notation, although it's not unusual to just use unmarked lines to connect tables and infer their relationship from the primary and foreign keys. Microsoft Access's "Relationships" tool can also be used to draw schema diagrams.

Primary key columns are underlined in the schema notation (as were identifier attributes) and may also be followed by "`(PK)`". Foreign key columns are indicated by "`(FK`*x*`)`" where *x* is a number assigned to the foreign key to distinguish it from other foreign keys in the table. Columns that accept NULL values are identified by "`: NULL`" following the column name. All other columns are assumed not to accept NULL values.

3.5 Conversion Specifics

The actual mechanics of converting entities into tables is relatively straightforward. Each of the special cases is described in the following sections.

3.5.1 One-to-One Relationships

One-to-one relationships are recognized in tables by inserting a foreign key column or columns into one of the tables that refers to the primary key of the other table. The inserted foreign key becomes the primary key of that table. Which table will receive the foreign key is a design decision, but if one already has a primary key, then the other table should add the foreign key column(s), as in this example:

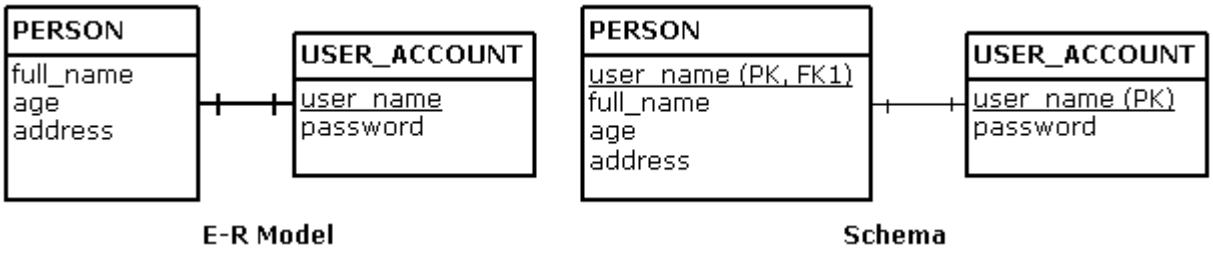


Figure 4.3 One-to-One Relationships
Source: http://college.yukondude.com/

Since USER_ACCOUNT already had a primary key (from the user_name identifier), then PERSON will receive the user_name foreign key column and it will become the primary key for the PERSON table.

3.5.2 One-to-Many Relationships

A one-to-many relationship is converted by inserting a foreign key into the table that lies on the "many" side of the relationship.

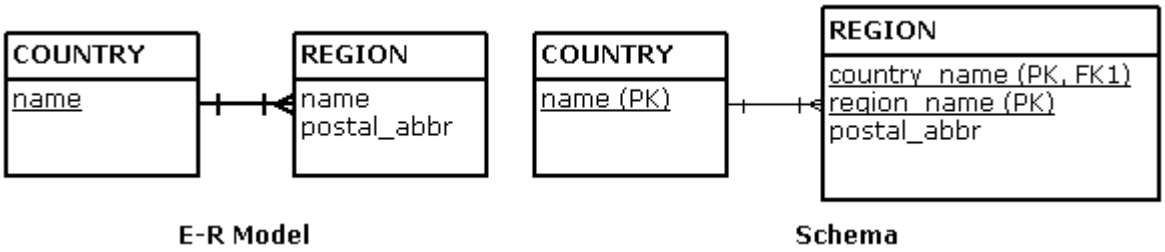


Figure 4.4 One-to-Many Relationships
Source: http://college.yukondude.com/

In this example, the foreign key is inserted into the REGION table because it lies on the "many" side of the relationship. Since REGION already had a column named "name", it was renamed to region_name and the foreign key was renamed to country_name so there wouldn't be any confusion. The primary key for REGION can now be chosen as the combination of country_name and region_name because both together guarantee uniqueness (a country will not have two regions with the same name).

3.5.3 Many-to-Many Relationships

To implement a many-to-many relationship in a schema, an additional table must be added that contains foreign keys for each of the two related tables. The additional table--called a **join table** (a.k.a. mapping, pivot, or junction table)--may also have additional columns for any attributes of the many-to-many relationship. The primary key of the new join table is the complete set of all foreign key columns.

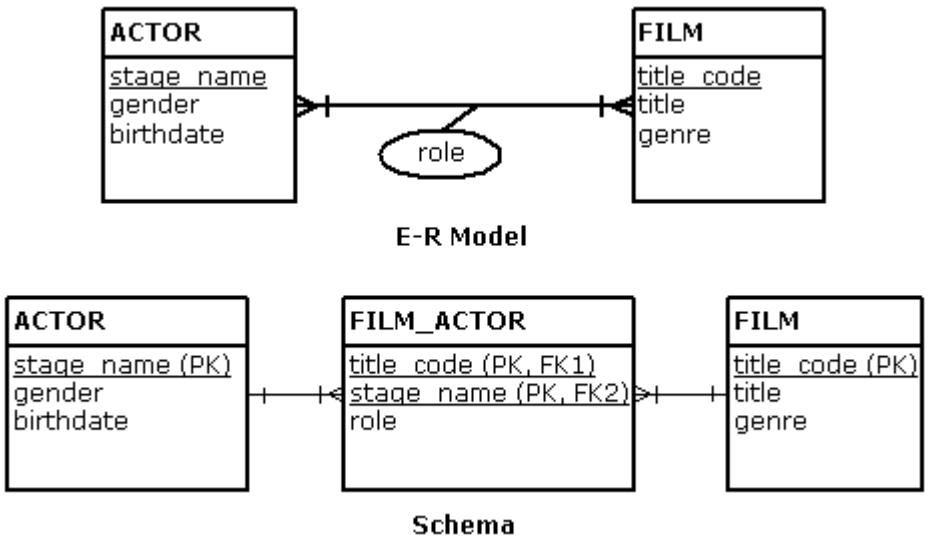


Figure 4.5 Many-to-Many Relationships
Source: http://college.yukondude.com/

In this example, both the ACTOR and FILM entities had unique identifiers, so those are chosen as the primary keys and are received by the new join table, FILM_ACTOR, as foreign keys. The role attribute of the many-to-many relationship becomes an attribute of the join table. The primary key for the FILM_ACTOR table is the combination of all foreign key columns.

If no other name suggests itself for the new join table, it's acceptable to concatenate the names of the two formerly-related tables as above (although ROLE might have made more sense--there's no reason a table can't have a column with the same name).

The "relationships" that connect the join table to the other tables will always be one-to-many, with the "many" side closest to the join table. Imagine slicing the many-to-many

relationship in half, and then swapping the two halves and placing them on either side of the join table.

Activity A

1. Explain the following terms
   i. Candidate Key
   ii. Foreign Key
   iii. Surrogate Key
   iv. Null value

2. Explain with aid of diagram how the following E-R notation can be converted to Relational Model. Pay good attention to Primary, Foreign and Surrogate keys.

3.6 Relationship Participation

The participation of a relationship (optional or mandatory) is converted to schema form depending upon the location of the foreign keys. The rule depends on the location of the relationship's optional symbol in the E-R diagram in relation to the table that receives the foreign key in the schema:

- If the optional symbol is on the *same side* as the foreign key, no further action is necessary.
- If the optional symbols is on the *opposite side* from the foreign key, the foreign key column(s) must accept NULL values.

For example, the optional symbol is on the same side as the foreign key in this model/schema ("a country may not have any sub-regions"):

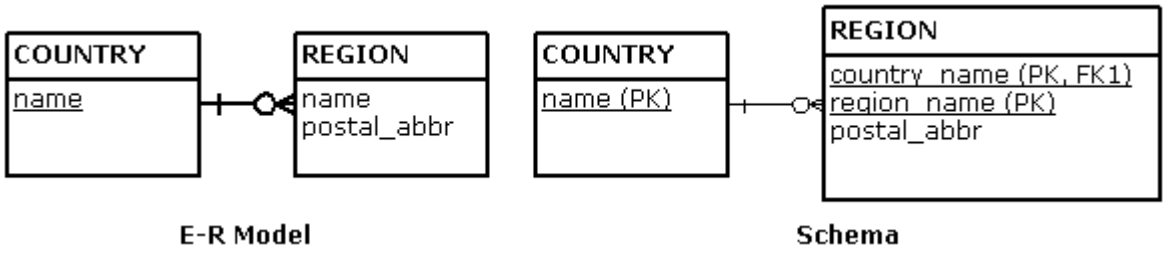


Figure 4.6(a) Relationships Participation
Source: http://college.yukondude.com/

The resulting schema is exactly the same as it would have been for a full mandatory relationship.

In this next example, the optional symbol is on the opposite side from the foreign key ("some extra-country regions exist"):

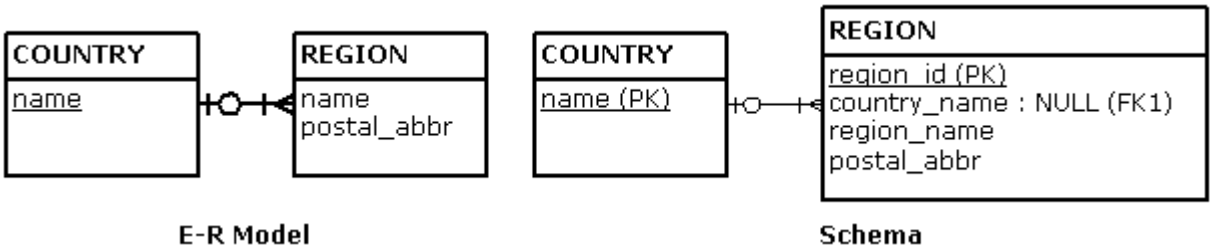


Figure 4.6(b) Relationships Participation
Source: http://college.yukondude.com/

The converted schema is considerably different from the previous example:

i. The foreign key still goes on the "many" side of the one-to-many relationship, and, as before, it is renamed country_name to avoid confusion.
ii. Because this foreign key is on the opposite side from the optional symbol, its column must accept NULL values.
iii. A primary key cannot contain columns that accept NULL values, so the combination of country_name and region_name no longer services.
iv. A new surrogate key, region_id, is instead chosen as the REGION table's primary key.

Now you see why we try to avoid optional participation when modelling. It can make quite a difference to a schema, but only when the optional symbol appears on the "one" side of a relationship. (Therefore, many-to-many relationships are immune from this complexity--join tables never contain NULL foreign key columns.)

3.7 Subtype Entities

At the relational schema level, a subtype relationship is nothing more than a one-to-one relationship that is optional on the side of the subtype table.

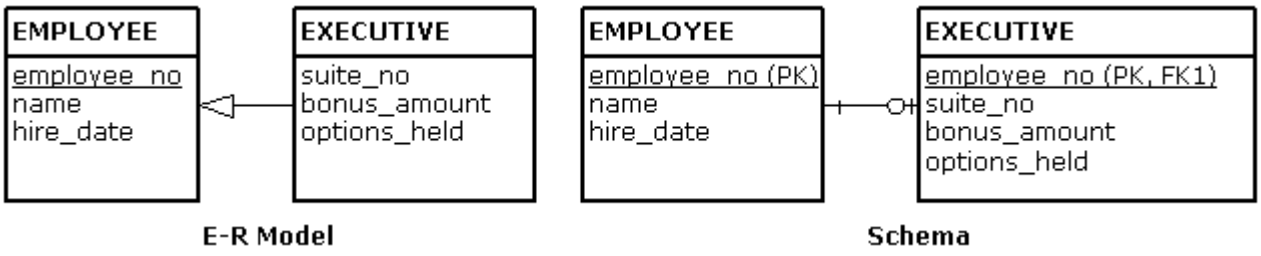


Figure 4.7 Subtype Entities Schema
Source: http://college.yukondude.com/

In this example, EXECUTIVE is a subtype of EMPLOYEE, and has extra attributes that apply only to executive employees. Once converted to schema form, the relationship is nothing more than one-to-one. A subtype table will always carry a foreign key to the supertype or parent table, and that foreign key will almost always serve as the primary key for the subtype table. Because the foreign key is on the same side of the relationship as the optional symbol, it does not accept NULL values.

The same principle applies to all non-subtype one-to-one relationships with one side that has an optional participation: whenever possible, put the foreign key into the table on the same side as the optional to avoid NULL foreign key columns.

Note that the "is a" sense of the subtype relationship evaporates after schema conversion. The EXECUTIVE table doesn't automatically inherit EMPLOYEE's attributes; the association between the two tables is now indistinguishable from a normal one-to-one relationship.

3.8 Reflexive Relationships

Finally, reflexive relationships are really no different than inter-entity relationships. Foreign keys are allocated using the same rules, and extra join tables are added for many-to-many reflexive relationships. The only difference is that the foreign keys for one-to-one or one-to-many reflexive relationships will end up in the same table, and so must be renamed.

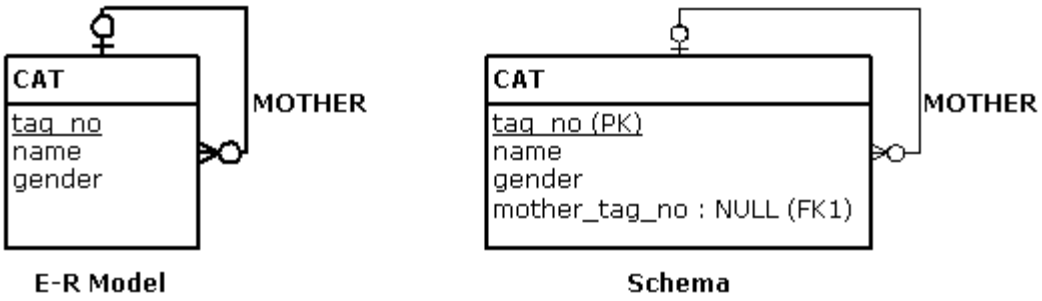


Figure 4.8 Reflexive Relationships
Source: http://college.yukondude.com/

In this example, the reflexive relationship represents cat mothers: "a cat may have many kittens, or none if it is not a mother, and a cat must have one mother, or none at all if the mother is unknown." Because this is a one-to-many relationship, the foreign key goes on the "many" side of the relationship, or right back into the CAT table. There is already a

tag_no column, so the foreign key must be renamed ("mother_tag_no" seems reasonable because it mentions the reflexive relationship's label).

It may not be immediately obvious, but this foreign key is on the opposite side from the optional symbol that appears on the "one" side of the reflexive relationship. Therefore the mother_tag_no column must accept NULL values.

3.9 Properties of Relational Tables

Relational tables have six properties:

i. Values are atomic.
ii. Column values are of the same kind.
iii. Each row is unique.
iv. The sequence of columns is insignificant.
v. The sequence of rows is insignificant.
vi. Each column must have a unique name.

(i) Values Are Atomic

This property implies that columns in a relational table are not repeating group or arrays. Such tables are referred to as being in the "first normal form" (1NF). The atomic value property of relational tables is important because it is one of the cornerstones of the relational model.

(ii) Column Values are of the Same Kind

In relational terms this means that all values in a column come from the same domain. A domain is a set of values which a column may have. For example, a Monthly_Salary column contains only specific monthly salaries. It never contains other information such as comments, status flags, or even weekly salary.

This property simplifies data access because developers and users can be certain of the type of data contained in a given column. It also simplifies data validation. Because all values are from the same domain, the domain can be defined and enforced with the Data Definition Language (DDL) of the database software.

(iii) Each Row is Unique

This property ensures that no two rows in a relational table are identical; there is at least one column, or set of columns, the values of which uniquely identify each row in the table. Such columns are called primary keys and are discussed in more detail in

(iv) The Sequence of Columns is Insignificant

This property states that the ordering of the columns in the relational table has no meaning. Columns can be retrieved in any order and in various sequences. The benefit of this property is that it enables many users to share the same table without concern of how the table is organized. It also permits the physical structure of the database to change without affecting the relational tables.

(v) The Sequence of Rows is Insignificant

This property is analogous the one above but applies to rows instead of columns. The main benefit is that the rows of a relational table can be retrieved in different order and sequences. Adding information to a relational table is simplified and does not affect existing queries.

(vi) Each Column has a Unique Name

Because the sequence of columns is insignificant, columns must be referenced by name and not by position. In general, a column name need not be unique within an entire database but only within the table to which it belongs.

3.10 Relational Data Integrity

Data integrity means, in part, that you can correctly and consistently navigate and manipulate the tables in the database. There are two basic rules to ensure data integrity; entity integrity and referential integrity.

The *entity integrity* rule states that the value of the primary key can never be a null value (a null value is one that has no value and is not the same as a blank). Because a primary key is used to identify a unique row in a relational table, its value must always be specified and should never be unknown. The integrity rule requires that insert, update, and delete operations maintain the uniqueness and existence of all primary keys.

The *referential integrity rule* states that if a relational table has a foreign key, then every value of the foreign key must either be null or match the values in the relational table in which that foreign key is a primary key.

Activity B

1. List six properties of Relational Tables
2. Explain the following terms
   i. Reflexive Relationships
   ii. Entity Integrity
   iii. Referential Integrity

4.0 Conclusion

A basic understanding of the relational model is necessary to effectively use relational database software such as Oracle, Microsoft SQL Server, or even personal database systems such as Access or Fox, which are based on the relational model.

5.0 Summary

In this unit, we have learnt:

vii. Conversion from E-R Model to RM and the rules governing the conversion
viii. NULL values are most often used in columns to indicate either Unknown or Not Applicable
ix. Keys are very important to relational schemas; much more so than identifiers to E-R models.
x. A one-to-many relationship is converted by inserting a foreign key into the table that lies on the "many" side of the relationship.
xi. To implement a many-to-many relationship in a schema, an additional table must be added that contains foreign keys for each of the two related tables.
xii. Relational tables have six properties: Values are atomic; Column values are of the same kind; Each row is unique; The sequence of columns is insignificant; The sequence of rows is insignificant; and Each column must have a unique name.
xiii. There are two basic rules to ensure data integrity; entity integrity and referential integrity.
xiv. Relational tables are sets. The rows of the tables can be considered as elements of the set. Operations that can be performed on sets can be done on relational tables. The eight relational operations are: Union; Product; Division; Projection; Join; Selection; Intersection; and Difference

6.0 Tutor Marked Assignment

1. Convert the student grading system E-R model generated in unit 3 TMA to their relational schema representation. When converting, pay special attention to: primary keys, foreign keys, many-to-many relationships, and Nulls. *At this stage of the game, try to avoid surrogate keys if at all possible.*

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 1: Database Management systems Concepts

Unit 5: Data Redundancy and Normalization

1.0 Introduction

Normalization is a design technique that is widely used as a guide in designing relational databases. Normalization is essentially a two step process that puts data into tabular form by removing repeating groups and then removes duplicated data from the relational tables.

Normalization theory is based on the concepts of **normal forms**. A relational table is said to be a particular normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined. In this unit, we will cover the first three normal forms that were defined by E. F. Codd.

2.0 Objectives

By the end of this unit, you should be able to:

f. Explain Functional Depencencies
g. Explain Data redundancy and Modification Anomalies
h. Understand what Normalization is all about
i. Explain First, Second, Third, Fourth and Fifth normal forms
j. Know when to de Normalize your Tables

3.1 Data Redundancy

Data Redundancy is a condition that exists when a data environment contains unnecessarily duplicated data. A change or modification, to redundant data, requires that you make changes to multiple fields of a database. While this is the expected behaviour for flat file database designs and spreadsheets, it defeats the purpose of relational database designs. The data relationships, inherent in a relational database, should allow you to maintain a single data field, at one location, and make the database's relational model responsible to port any changes, to that data field, across the database. Redundant data wastes valuable space and creates troubling database maintenance problems. Data Redundancy is the bane of DBMS development and management as it wastes resources.

Unnecessary data can occur when an organization stores several copies of similar information about the same data in multiple departments within an organization (ie; Sales, Support, and Marketing) maintaining their "own" customer databases (ie; SALES_CUST, SUPPORT_CUST, and MARKETING_CUST). It can also occur if repeatable data types are contained within repeating fields, and not segregated into their own tables and related by a unique ID key (ie; CUST_ID).

3.1.1 Reasons against most types of *unnecessary* duplicate data:

- Change requires reconciling data in multiple locations or fields in the database
- Wastes physical storage space
- Can decrease performance, accuracy, reliability, and stability
- Introduces maintenance issues
- Increases difficulty of troubleshooting database problems
- Data redundancy results in data inconsistency, or a lack of data integrity.
  - Different and conflicting versions of the same data appear in different places
- Errors more likely to occur when complex entries are made in several different files and/or recur frequently in one or more files
- Data anomalies develop when required reconciliations in redundant data are not made successfully

3.1.2 Types of data anomalies:

- Update anomalies
  - Occur when changes must be made to existing records
- Insertion anomalies
  - Occur when entering new records
- Deletion anomalies
  - Occur when deleting records

3.1.3 How to Eliminate Redundant Data:

To eliminate redundant data from your database, you must take special care to organize the data in your data tables. Normalization is a method of organizing your data to prevent redundancy. Normalization involves establishing and maintaining the integrity of your data tables as well as eliminating inconsistent data dependencies.

Normalization requires that you adhere to rules, established by the database community, to ensure that data is organized efficiently. These rules are called normal form rules. Normalization may require that you include additional data tables in your database. Normal form rules number from one to three, for most applications. The rules are cumulative such that the rules of the 2nd normal form are inclusive of the rules in the 1st normal form. The rules of the 3rd normal form are inclusive of the rules in the 1st and 2nd normal forms, etc.

3.2 Basic Concepts of Normalization

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a

relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are:

- mutually independent and
- fully dependent upon the primary key.

Mutual independence means that no non-key column is dependent upon any combination of the other columns. The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF. In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of functional dependencies and lossless decomposition.

3.3 Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms. A column, Y, of the relational table R is said to be **functionally dependent** upon column X of R if and only if each value of X in R is associated with precisely one value of Y at any given time. X and Y may be composite. Saying that column Y is functionally dependent upon X is the same as saying the values of column X identify the values of column Y. If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

A short-hand notation for describing a functional dependency is:

R.x —>; R.y

which can be read as in the relational table named R, column x functionally determines (identifies) column y.

Full functional dependence applies to tables with composite keys. Column Y in relational table R is fully functional on X of R if it is functionally dependent on X and not functionally dependent upon any subset of X. Full functional dependence means that when a primary key is composite, made of two or more columns, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

3.4 Overview of Normalization

Simply stated, normalization is the process of removing redundant data from relational tables by decomposing (splitting) a relational table into smaller tables by projection. The goal is to have only primary keys on the left hand side of a functional dependency. In order to be correct, decomposition must be lossless. That is, the new tables can be recombined by a natural join to recreate the original table without creating any spurious or redundant data.

3.5 Sample Data

A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information that can be expressed in relational notation as:

FIRST (s#, status, city, p#, qty)

where

s#	supplier identifcation number (this is the primary key)
status	status code assigned to city
city	name of city where supplier is located
p#	part number of part supplied
qty>	quantity of parts supplied to date

In order to uniquely associate quantity supplied (qty) with part (p#) and supplier (s#), a composite primary key composed of s# and p# is used.

3.6 First Normal Form

A relational table, by definition, is in first normal form if all values of the columns are atomic. That is, they contain no repeating values. Figure1 shows the table FIRST in 1NF.

FIRST

s#	status	city	p#	qty
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

Figure 5.1: Table in 1NF
Source: http://www.utexas.edu/

Although the table FIRST is in 1NF it contains redundant data. For example, information about the supplier's location and the location's status has to be repeated for every part supplied. Redundancy causes what are called update anomalies. Update anomalies are problems that arise when information is inserted, deleted, or updated. For example, the following anomalies could occur in FIRST:

- INSERT. The fact that a certain supplier (s5) is located in a particular city (Athens) cannot be added until they supplied a part.
- DELETE. If a row is deleted, then not only is the information about quantity and part lost but also information about the supplier.
- UPDATE. If supplier s1 moved from London to New York, then six rows would have to be updated with this new information.

3.7 Second Normal Form

The definition of second normal form states that only tables with composite primary keys can be in 1NF but not in 2NF.

> A relational table is in second normal form 2NF if it is in 1NF and every non-key column is fully dependent upon the primary key.

That is, every non-key column must be dependent upon the entire primary key. FIRST is in 1NF but not in 2NF because status and city are functionally dependent upon only on the column s# of the composite key (s#, p#). This can be illustrated by listing the functional dependencies in the table:

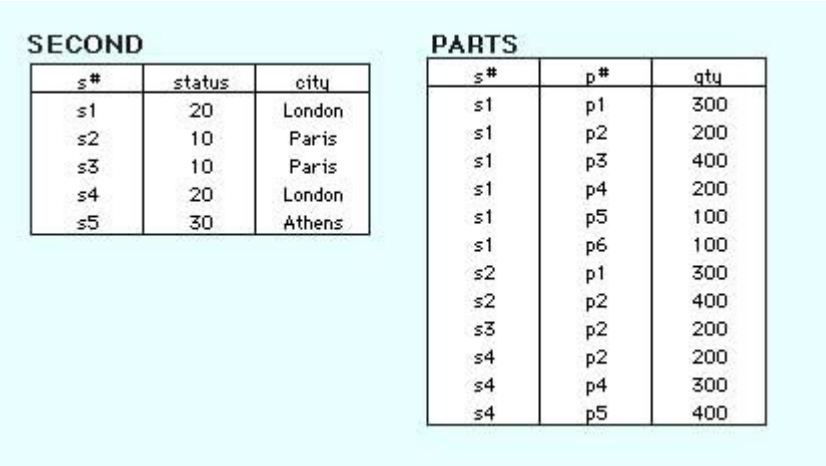
$$s\# \longrightarrow city, status$$
$$city \longrightarrow status$$
$$(s\#,p\#) \longrightarrow qty$$

The process for transforming a 1NF table to 2NF is:

1. Identify any determinants other than the composite key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.
3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.

4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform FIRST into 2NF we move the columns s#, status, and city to a new table called SECOND. The column s# becomes the primary key of this new table. The results are shown below in Figure 2.



SECOND

s#	status	city
s1	20	London
s2	10	Paris
s3	10	Paris
s4	20	London
s5	30	Athens

PARTS

s#	p#	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Figure 5.2: Tables in 2NF
Source: http://www.utexas.edu/

Tables in 2NF but not in 3NF still contain modification anomalies. In the example of SECOND, they are:

> INSERT. The fact that a particular city has a certain status (Rome has a status of 50) cannot be inserted until there is a supplier in the city.
>
> DELETE. Deleting any row in SUPPLIER destroys the status information about the city as well as the association between supplier and city.

3.8 Third Normal Form

The third normal form requires that all columns in a relational table are dependent only upon the primary key. A more formal definition is:

> A relational table is in third normal form (3NF) if it is already in 2NF and every non-key column is non transitively dependent upon its primary key. In other words, all nonkey attributes are functionally dependent only upon the primary key.

Table PARTS is already in 3NF. The non-key column, qty, is fully dependent upon the primary key (s#, p#). SUPPLIER is in 2NF but not in 3NF because it contains a transitive dependency. A transitive dependency is occurs when a non-key column that is a determinant of the primary key is the determinate of other columns. The concept of a transitive dependency can be illustrated by showing the functional dependencies in SUPPLIER:

SUPPLIER.s# —> SUPPLIER.status

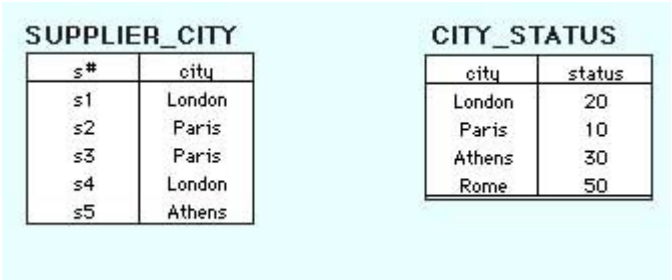
SUPPLIER.s# —> SUPPLIER.city

SUPPLIER.city —> SUPPLIER.status

Note that SUPPLIER.status is determined both by the primary key s# and the non-key column city. The process of transforming a table into 3NF is:

1. Identify any determinants, other the primary key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.
3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform SUPPLIER into 3NF, we create a new table called CITY_STATUS and move the columns city and status into it. Status is deleted from the original table, city is left behind to serve as a foreign key to CITY_STATUS, and the original table is renamed to SUPPLIER_CITY to reflect its semantic meaning. The results are shown in Figure 3 below.



SUPPLIER_CITY

s#	city
s1	London
s2	Paris
s3	Paris
s4	London
s5	Athens

CITY_STATUS

city	status
London	20
Paris	10
Athens	30
Rome	50

Figure 5.3: Tables in 3NF
Source: http://www.utexas.edu/

The results of putting the original table into 3NF has created three tables. These can be represented in "psuedo-SQL" as:

PARTS (#s, p#, qty)
Primary Key (s#,#p)
Foreign Key (s#) references SUPPLIER_CITY.s#

SUPPLIER_CITY(s#, city)
Primary Key (s#)
Foreign Key (city) references CITY_STATUS.city

CITY_STATUS (city, status)
Primary Key (city)

3.8.1 Advantages of Third Normal Form

The advantage of having relational tables in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies. For example, the improvements to our sample database are:

> INSERT. Facts about the status of a city, Rome has a status of 50, can be added even though there is not supplier in that city. Likewise, facts about new suppliers can be added even though they have not yet supplied parts.
>
> DELETE. Information about parts supplied can be deleted without destroying information about a supplier or a city. UPDATE. Changing the location of a supplier or the status of a city requires modifying only one row.

Activity A

1. Explain the following terms
   i. Functional dependency
   ii. First, Second, and Third Normal Forms

2. What do you understand by modification anomalies?

3.9 Advanced Normalization

After 3NF, all normalization problems involve only tables which have three or more columns and all the columns are keys. Many practitioners argue that placing entities in 3NF is generally sufficient because it is rare that entities that are in 3NF are not also in 4NF and 5NF. They further argue that the benefits gained from transforming entities into 4NF and 5NF are so slight that it is not worth the effort. However, advanced normal forms are presented because there are cases where they are required.

3.9.1 Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) is a more rigorous version of the 3NF deal with relational tables that had

(a) multiple candidate keys,
(b) composite candidate keys, and
(c) candidate keys that overlapped .

BCNF is based on the concept of determinants. A determinant column is one on which some of the columns are fully functionally dependent.

A relational table is in BCNF if and only if every determinant is a candidate key.

3.9.2 Fourth Normal Form

A relational table is in the fourth normal form (4NF) if it is in BCNF and all multivalued dependencies are also functional dependencies.

Fourth normal form (4NF) is based on the concept of multivalued dependencies (MVD). A Multivalued dependency occurs when in a relational table containing at least three columns, one column has multiple rows whose values match a value of a single row of one of the other columns. A more formal definition given by Date is:

given a relational table R with columns A, B, and C then

R.A —>> R.B (column A multidetermines column B)

is true if and only if the set of B-values matching a given pair of A-values and C-values in R depends only on the A-value and is independent of the C-value.

MVD always occur in pairs. That is R.A —>> R.B holds if and only if R.A —>> R.C also holds.

Suppose that employees can be assigned to multiple projects. Also suppose that employees can have multiple job skills. If we record this information in a single table, all three attributes must be used as the key since no single attribute can uniquely identify an instance.

The relationship between emp# and prj# is a multivalued dependency because for each pair of emp#/skill values in the table, the associated set of prj# values is determined only by emp# and is independent of skill. The relationship between emp# and skill is also a multivalued dependency, since the set of Skill values for an emp#/prj# pair is always dependent upon emp# only.

To transform a table with multivalued dependencies into the 4NF move each MVD pair to a new table. The result is shown in Figure1.

Figure 5.4: Tables in 4NF
Source: *http://www.utexas.edu/*

3.9.3 Fifth Normal Form

A table is in the fifth normal form (5NF) if it cannot have a lossless decomposition into any number of smaller tables.

While the first four normal forms are based on the concept of functional dependence, the fifth normal form is based on the concept of join dependence. Join dependency means that an table, after it has been decomposed into three or more smaller tables, must be capable of being joined again on common keys to form the original table. Stated another way, 5NF indicates when an entity cannot be further decomposed. 5NF is complex and not intuitive. Most experts agree that tables that are in the 4NF are also in 5NF except for "pathological" cases. Teorey suggests that true many-to-many-to-many ternary relations are one such case.

Adding an instance to an table that is not in 5NF creates spurious results when the tables are decomposed and then rejoined. For example, let's suppose that we have an employee who uses design skills on one project and programming skills on another. This information is shown below.

emp#	prj#	skill

1211	11	Design
1211	28	Program

Next we add an employee (1544) who uses programming skills on Project 11.

emp#	prj#	skill
1211	11	Design
1211	28	Program
1544	11	Program

Next, we project this information into three tables as we did above. However, when we rejoin the tables, the recombined table contains spurious results.

emp#	prj#	skill	
1211	11	Design	
1211	11	Program	<<—spurious data
1211	28	Program	
1544	11	Design	<<—spurious data
1544	11	Program	

By adding one new instance to a table not in 5NF, two false assertions were stated:

Assertion 1

- Employee 1211 has been assigned to Project 11.
- Project 11 requires programming skills.
- Therefore, Employee 1211 must use programming skills while assigned to Project 11.

Assertion 2

- Employee 1544 has been assigned to project 11.
- Project 11 needs Design skills.
- Therefore, Employee 1544 must use Design skills in Project 11.

3.9.4 Domain Key Normal Form (DK/NF)

A relation is in DK/NF if every *constraint* on the relation is a logical consequence of the definition of *keys* and *domains*.

Constraint: An rule governing static values of an attribute such that we can determine if this constraint is True or False. Examples:

i. Functional Dependencies
ii. Multivalued Dependencies
iii. Inter-relation rules
iv. Intra-relation rules

However: Does *not* include time dependent constraints.

Key: Unique identifier of a tuple.

Domain: The physical (data type, size, NULL values) and semantic (logical) description of what values an attribute can hold.

There is no known algorithm for converting a relation directly into DK/NF.

3.10 De-Normalization

Consider the following relation:
CUSTOMER (CustomerID, Name, Address, City, State, Zip)

This relation is not in DK/NF because it contains a functional dependency not implied by the key.

```
Zip -> City, State
```

We can normalize this into DK/NF by splitting the CUSTOMER relation into two:
CUSTOMER (CustomerID, Name, Address, Zip)
CODES (Zip, City, State)

We may pay a performance penalty - each customer address lookup requires we look in two relations (tables).

In such cases, we may *de-normalize* the relations to achieve a performance improvement.

3.11 Example

This is an example that runs through all of the normal forms from beginning to end using the same tables.

Example relation:
EMPLOYEE (Name, Project, Task, Office, Phone)

Note: Keys are underlined.

Example Data:

Name	Project	Task	Office	Floor	Phone
Bili	100X	T1	400	4	1400
Bili	100X	T2	400	4	1400
Bili	200Y	T1	400	4	1400
Bili	200Y	T2	400	4	1400
Sule	100X	T33	442	4	1442
Sule	200Y	T33	442	4	1442
Sule	300Z	T33	442	4	1442
Edo	100X	T2	588	5	1588

- **Name** is the employee's name
- **Project** is the project they are working on. Bill is working on two different projects, Sue is working on 3.
- **Task** is the current task being worked on. Bill is now working on Tasks T1 and T2. Note that Tasks are independent of the project. Examples of a task might be faxing a memo or holding a meeting.
- **Office** is the office number for the employee. Bill works in office number 400.
- **Floor** is the floor on which the office is located.
- **Phone** is the phone extension. Note this is associated with the phone in the given office.

First Normal Form

Assume the **key** is Name, Project, Task.

Is EMPLOYEE in 1NF?

Second Normal Form

List all of the functional dependencies for EMPLOYEE.

Are all of the non-key attributes dependant on all of the key?

Split into two relations EMPLOYEE_PROJECT_TASK and EMPLOYEE_OFFICE_PHONE. EMPLOYEE_PROJECT_TASK (Name, Project, Task)

Name	Project	Task
Bili	100X	T1
Bili	100X	T2
Bili	200Y	T1
Bili	200Y	T2
Sule	100X	T33
Sule	200Y	T33
Sule	300Z	T33
Edo	100X	T2

EMPLOYEE_OFFICE_PHONE (Name, Office, Floor, Phone)

Name	Office	Floor	Phone
Bili	400	4	1400
Sule	442	4	1442
Edo	588	5	1588

Third Normal Form

Assume each office has exactly one phone number.

Are there any transitive dependencies?

Where are the modification anomalies in EMPLOYEE_OFFICE_PHONE?

Split EMPLOYEE_OFFICE_PHONE.

EMPLOYEE_PROJECT_TASK (Name, Project, Task)

Name	Project	Task
Bili	100X	T1
Bili	100X	T2
Bili	200Y	T1

Bili	200Y	T2
Sule	100X	T33
Sule	200Y	T33
Sule	300Z	T33
Edo	100X	T2

EMPLOYEE_OFFICE (Name, Office, Floor)

Name	**Office**	**Floor**
Bili	400	4
Sule	442	4
Edo	588	5

EMPLOYEE_PHONE (Office, Phone)

Office	**Phone**
400	1400
442	1442
588	1588

Boyce-Codd Normal Form

List all of the functional dependencies for EMPLOYEE_PROJECT_TASK, EMPLOYEE_OFFICE and EMPLOYEE_PHONE. Look at the determinants.

Are all determinants candidate keys ?

Forth Normal Form

Are there any multivalued dependencies ?

What are the modification anomalies ?

Split EMPLOYEE_PROJECT_TASK.

EMPLOYEE_PROJECT (Name, Project)

Name	**Project**

Bili	100X
Bili	200Y
Sule	100X
Sule	200Y
Sule	300Z
Edo	100X

EMPLOYEE_TASK (Name, Task)

Name	**Task**
Bili	T1
Bili	T2
Sule	T33
Edo	T2

EMPLOYEE_OFFICE (Name, Office, Floor)

Name	**Office**	**Floor**
Bili	400	4
Sule	442	4
Edo	588	5

R4 (Office, Phone)

Office	**Phone**
400	1400
442	1442
588	1588

At each step of the process, we did the following:

1. Write out the relation
2. (optionally) Write out some example data.
3. Write out all of the functional dependencies

4. Starting with 1NF, go through each normal form and state why the relation is in the given normal form.

Activity B

1. Explain the following terms:
   i. Boyce-Codd Normal Form
   ii. Fourth Normal Form
   iii. Fifth Normal Form

2. What are the benefits of Normalization?

4.0 Conclusion

Once our E-R model has been converted into relations, we may find that some relations are not properly specified. There can be a number of problems:

- **Deletion Anomaly**: Deleting a relation results in some related information (from another entity) being lost.
- **Insertion Anomaly**: Inserting a relation requires we have information from two or more entities - this situation might not be feasible.

Typical way to solve these anomalies is to split the relation in to two or more relations - Process called *Normalization*.

5.0 Summary

In this unit, we have learnt:

xv. Data Redundancy is a condition that exists when a data environment contains unnecessarily duplicated data.
xvi. Normalization is a set of techniques for organizing data into tables in such a way to eliminate certain type of redundancy and incompleteness, and associated complexity and/or anomalies when updating it.
xvii. Normalization is a set of techniques for organizing data into tables in such a way to eliminate certain type of redundancy and incompleteness, and associated complexity and/or anomalies when updating it.
xviii.
xix. The designer starts with a single file and divides it into tables based on dependencies among the data item.
xx. Normalization relies on correct identification of determinants and keys.
xxi. A table is in 3NF if every determinant of a non key item is a candidate key.
xxii. In practice, normalization is used primarily as a check on the correctness of a model developed using a top-down approach.

6.0 Tutor Marked Assignment

Question 1

A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information that can be expressed in relational notation as:

FIRST (s#, status, city, p#, qty)

where

s# supplier identifcation number (this is the primary key)
status status code assigned to city
city name of city where supplier is located
p# part number of part supplied
qty> quantity of parts supplied to date

In order to uniquely associate quantity supplied (qty) with part (p#) and supplier (s#), a composite primary key composed of s# and p# is used.

The sample data is shown below:

FIRST

s#	status	city	p#	qty
s1	10	Lagos	p1	300
s1	10	Lagos	p2	200
s1	10	Lagos	p3	400
s1	10	Lagos	p4	200
s1	10	Lagos	p5	100
s1	10	Lagos	p6	100
s2	15	Jos	p1	300
s2	15	Jos	p2	400
s3	15	Jos	p2	200
s4	10	Lagos	p2	200
s4	10	Lagos	p4	300
s4	10	Lagos	p5	400

Use the above information to explain the following in detail

a. Normalization
b. Functional Dependencies
c. First Normal Form

d. Update anomalies
e. Second Normal Form
f. Third Normal Form

7.0 Further Reading and other Resources

databasedev.co.uk (2003-2006). *Data Redundancy Defined - Relational Database Design.* In, Database Solutions for Microsoft Access, Retrieved October 10th, 2006, from: http://www.databasedev.co.uk/data-redundancy.html

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 1: Database Management systems Concepts

Unit 6: Relational Algebra

1.0 Introduction

Relational tables are sets. The rows of the tables can be considered as elements of the set. Operations that can be performed on sets can be done on relational tables. The relational operations are the main focus of this unit:

2.0 Objectives

By the end of this unit, you should be able to:

k. Know what relational algebra is all about
l. Have good understanding of Set Theory operations which include: Union, Intersection, Difference and Cartesian product.
m. Know how to use Specific Relational Operations: Selection, Projection, Join, and Division in a relation.

3.0 Relational Operations

Let us recall that,

a. The Relational Model consists of the elements: relations, which are made up of attributes.
b. A relation is a set of attributes with values for each attribute.
c. Relational Algebra is a collection of operations on Relations.
d. Relations are operands and the result of an operation is another relation.
e. Two main collections of relational operators:

    i. Set theory operations:
    Union, Intersection, Difference and Cartesian product.
    ii. Specific Relational Operations:
    Selection, Projection, Join, Division

3.1 Set Theoretic Operations

In this section, we shall consider the following set operations: Union, Intersection, and Difference.

Consider the following relations **A** and **B**

A

Firstname	Suurname	Score
John	Ayodeji	75

Sheu	Abdul	76
Taiwo	Lawrence	77
Dan	Musa	78

B

Firstname	Surname	Score
Wale	Osa	65
Chidi	Brown	66
Dan	Musa	78

3.1.1 Union: A ∪B

The *union* operation of two relational tables is formed by appending rows from one table to those of a second table to produce a third. Duplicate rows are eliminated.

A ∪B

Firstname	Suurname	Score
John	Ayodeji	75
Sheu	Abdul	76
Taiwo	Lawrence	77
Dan	Musa	78
Wale	Osa	65
Chidi	Brown	66

3.1.2 Difference: A - B

The *difference* of two relational tables is a third that contains those rows that occur in the first table but not in the second. The Difference operation requires that the tables be union compatible. As with arithmetic, the order of subtraction matters. That is, A - B is not the same as B - A.

A - B

Firstname	Surname	Score
John	Ayodeji	75
Sheu	Abdul	76
Taiwo	Lawrence	77

3.1.3 Intersection: A ∩B

The intersection of two relational tables is a third table that contains common rows. Both tables must be union compatible.

A ∩B

Firstname	Surname	Score
Dan	Musa	78

3.2 Union Compatible Relations

- Attributes of relations need not be identical to perform union, intersection and difference operations.
- However, they must have the same number of attributes or arity and the domains for corresponding attributes must be identical.
- **Domain** is the data type and size of an attribute.
- The **degree** of relation R is the number of attributes it contains.
- Definition: Two relations R and S are *union compatible* if and only if they have the same degree and the domains of the corresponding attributes are the same.
- Some additional properties:
    - Union, Intersection and difference operators may only be applied to Union Compatible relations.
    - Union and Intersection are commutative operations
      R ∪S = S ∪R
      R ∩S = S ∩R
    - Difference operation is NOT commutative.
      R - S not equal S - R
    - The resulting relations may not have meaningful names for the attributes. Convention is to use the attribute names from the first relation.

Activity A

- Assume relation **C**

Firstame	Surname	Score
Sanyo	Kunbi	89
Alli	Barry	53
Dan	Musa	78

- Compute A $\cup$ C
  Compute A $\cap$ C
  Show that A - C is not equal to C - A

3.3 Cartesian Product

The Cartesian product of two relational tables is the concatenation of every row in one table with every row in the second. The product of table A (having m rows) and table B (having n rows) is the table C (having m x n rows). The product is denoted as A X B or A TIMES B.

A

Firstname	Suurname	Score
John	Ayodeji	75
Sheu	Abdul	76
Taiwo	Lawrence	77

B

Lunch	Drink
Pounded Yam	Malt
Bean Cake	Beer

A X B

Firstname	Suurname	Score	Lunch	Drink
John	Ayodeji	75	Pounded Yam	Malt
John	Ayodeji	75	Bean Cake	Beer
Sheu	Abdul	76	Pounded Yam	Malt
Sheu	Abdul	76	Bean Cake	Beer
Taiwo	Lawrence	77	Pounded Yam	Malt
Taiwo	Lawrence	77	Bean Cake	Beer

3.4 Selection and Projection Operations

The *project* operator retrieves a *subset of columns* from a table, removing duplicate rows from the result while the select operator, sometimes called restrict to prevent confusion with the SQL SELECT command, retrieves subsets of rows from a relational table based on a value(s) in a column or columns.

We shall consider these operators one after the other.

3.4.1 Selection Operators

The selection operator has the following characteristics:

i. Selection and Projection are *unary* operators.
ii. The selection operator is sigma: σ
iii. The selection operation acts like a *filter* on a relation by returning only a certain number of tuples.
iv. The resulting relation will have the same degree as the original relation.
v. The resulting relation may have fewer tuples than the original relation.
vi. The tuples to be returned are dependent on a *condition* that is part of the selection operator.
vii. $\sigma_{C}(R)$ Returns only those tuples in R that satisfy condition **C**
viii. A condition C can be made up of any combination of comparison or logical operators that operate on the attributes of R.
    - Comparison operators: $= \quad < \quad > \quad \geq \quad \leq \quad \neq$
    - Logical operators: $\wedge \quad \vee \quad \neg$
ix. Use the Truth tables (memorize these) for logical expressions:

$\wedge$	**T**	**F**
T	T	F
F	F	F

$\vee$	**T**	**F**
T	T	T
F	T	F

$\neg$	**T**	**F**
	F	T

3.4.2 Selection Examples

Let us assume that the following staff table (STAFF) has the following records:

Name	**Room**	**Department**	**Designation**
Ajayi	101	CIT	Professor
Chidi	201	ECO	Lecturer I
Musa	202	ECO	Professor
Bello	301	CIT	Senior Lecturer
Ajayi	220	ACC	Senior Lecturer

a. Select only those Staff in the CIT department:

$\sigma_{\text{Dept = 'CIT'}}$ (STAFF)

Result:

Name	**Room**	**Department**	**Designation**
Ajayi	101	CIT	Professor
Bello	301	CIT	Senior Lecturer

b. Select only those Staff with last name Ajayi who are professors:

$\sigma_{\text{Name = 'Ajayi'}} \wedge_{\text{Designation = 'Professor'}}$ (STAFF)

Result:

Name	Room	Department	Designation
Ajayi	101	CIT	Professor

c. Select only those Staff who are either Professors or in the Economics department:

$\sigma_{\text{Designation = 'Professor'} \vee \text{Department = 'ECO'}}(\text{STAFF})$

Result:

Name	Room	Department	Designation
Ajayi	101	CIT	Professor
Chidi	201	ECO	Lecturer I
Musa	202	ECO	Professor

d. Select only those Employees who are not in the CIT department or Lecturer I:

$\sigma_{\neg(\text{Designation = 'Lecturer I'} \vee \text{Department = 'CIT'})}(\text{STAFF})$

Result:

Name	Room	Department	Designation
Musa	202	ECO	Professor
Ajayi	220	ACC	Senior Lecturer

Activity B

1a. Evaluate the following expressions:

i. $\sigma_{\neg(\text{Designation = 'Lecturer I'} \wedge \text{Department = 'CIT'})}(\text{STAFF})$

ii. $\sigma_{\text{Designation = 'Professor'}}(\sigma_{\text{Department = 'CIT'}}\ \text{STAFF})$

iii. $\sigma_{\text{Department = 'CIT'}}(\sigma_{\text{Designation = 'Professor'}}\ \text{STAFF})$

iv. $\sigma_{\text{Designation = 'Professor'} \wedge \text{Department = 'CIT'}}(\text{STAFF})$

v. $\sigma_{\text{Score} > 60}(A \cup B)$

For this expression, use A and B from section 3.1.

b. Do expressions ii, iii and iv above all evaluate to the same thing?

3.4.3 Projection Operator

The projection operator has the following characteristics:

- Projection is also a Unary operator.
- The Projection operator is pi: π
- Projection limits the *attributes* that will be returned from the original relation.
- The general syntax is: $\pi_{attributes}$ R
  Where *attributes* is the list of attributes to be displayed and R is the relation.
- The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced).
- The degree of the resulting relation may be equal to or less than that of the original relation.

3.4.4 Projection Examples

Assume the same STAFF table above is used.

a. Project only the names and departments of the employees:

$\pi_{name,\ department}$ (STAFF)

Results:

Name	Department
Ajayi	CIT
Chidi	ECO
Musa	ECO
Bello	CIT
Ajayi	ACC

3.4.5 Combining Selection and Projection

The selection and projection operators can be combined to perform both operations.

a. Show the names of all employees working in the CS department:
π_{name} ($\sigma_{Department = 'CIT'}$ (STAFF))

Results:

Name
Ajayi
Bello

b. Show the name and designation of those Employees who are not in the CIT department or Lecturer I:

$\pi_{\text{name, Designation}} (\sigma \neg (\text{Designation = 'Lecturer I'} \vee \text{Department = 'CIT'}) (\text{STAFF}))$

Result:

Name	Designation
Musa	Professor
Ajayi	Senior Lecturer

Activity C

1. Evaluate the following expressions:

a. $\pi_{\text{name, Designation}} (\sigma \neg (\text{Designation = 'Lecturer I'} \wedge \text{Department = 'CIT'}) (\text{STAFF}))$

b. $\pi_{\text{Firstname, Score}} (\sigma_{\text{Score > 70}} (A \cup B))$

For this expression, use A and B from section 3.1.

c. $\sigma_{\text{room > 201}} (\pi_{\text{name, Designation}} (\text{STAFF}))$

3.5 Aggregate Functions

Aggregate functions are sometimes written using the *Projection* operator or the *Script F* character: $\mathcal{F}$ as in Elmasri/Navathe book. In this section, we shall consider the following aggregate functions:

- SUM
- MINIMUM
- MAXIMUM
- AVERAGE, MEAN, MEDIAN
- COUNT

3.5.1 Aggregate Function Examples

Let us assume that the table STAFF has the following records:

Name	Room	Department	Designation	Salary
Ajayi	101	CIT	Professor	45000
Chidi	201	ECO	Lecturer I	35000
Musa	202	ECO	Professor	50000
Bello	301	CIT	Senior Lecturer	65000
Ajayi	220	ACC	Senior Lecturer	60000

a. Find the minimum Salary: $\Im_{\text{MIN (salary)}}(\text{STAFF})$

Results:

MIN(salary)
35000

b. Find the average Salary: $\Im_{\text{AVG (salary)}}(\text{STAFF})$

Results:

AVG(salary)
51000

c. Count the number of employees in the CIT department: $\Im_{\text{COUNT (name)}}(\sigma_{\text{Department = 'CIT'}}(\text{STAFF}))$

Results:

COUNT(name)
2

d. Find the total payroll for the Economics department: $\Im_{SUM\ (salary)}$ ($\sigma_{Department = 'ECO'}$ (STAFF))

Results:

SUM(salary)
85000

3.6 Join Operation

A *join* operation combines the product, selection, and, possibly, projection. The join operator horizontally combines (concatenates) data from one row of a table with rows from another or the same table when certain criteria are met. The criteria involve a relationship among the columns in the join relational table. If the join criterion is based on equality of column value, the result is called an *equijoin*. A *natural join* is an equijoin with redundant columns removed. The following are the properties of join operation:

- Join operations bring together two tables and combine their columns and records or ows in a specific fashion.
- The generic join operator (called the *Theta Join* is: $\bowtie$
- It takes as arguments the columns from the two tables that are to be joined.
- For example assume we have the STAFF table as above and a separate DEPARTMENT table with (Dept, MainOffice, Phone) :
  STAFF $\bowtie_{STAFF.Department = DEPARTMENT.Dept}$ DEPARTMENT
- The join condition can be $= \quad < \quad > \quad \geq \quad \leq \quad \neq$
- When the join condition operator is = then we call this an *Equijoin*
- Note that the columns in common are repeated.

3.6.1 Join Examples

Let us assume we have the STAFF table from above and the following DEPARTMENT table:

Dept	MainOffice	Phone
CIT	404	010-234-0001
ECO	200	010-234-0002

ACC	501	010-234-0003
Yor	100	010-234-0004

a. Find all information on every employee including their department info:
STAFF $\bowtie_{staff.Department = department.Dept}$ DEPARTMENT

Results:

Name	Room	Department	Salary	Dept	MainOffice	Phone
Ajayi	101	CIT	45000	CIT	404	010-234-0001
Chidi	201	ECO	35000	ECO	200	010-234-0002
Musa	202	ECO	50000	ECO	200	010-234-0002
Bello	301	CIT	65000	ACC	501	010-234-0003
Ajayi	220	ACC	60000	Yor	100	010-234-0004

b. Find all information on every employee including their department info where the employee works in an office numbered less than the department main office:
STAFF $\bowtie_{(staff.room < department.mainoffice) \wedge (staff.department = department.dept)}$ DEPARTMENT

Results:

Name	Room	Department	Salary	Dept	MainOffice	Phone
Ajayi	101	CIT	45000	CIT	404	010-234-0001
Bello	301	CIT	65000	ACC	501	010-234-0003

4.0 Conclusion

The relational algebra is a procedural query language with several fundamental operations: select (unary), project (unary), rename (unary), cartesian product (binary), union (binary), set-difference (binary), set-intersection, natural join, division, assignment. Operations produce a new relation as a result.

5.0 Summary

In this unit, we have learnt:

xxiii. The *union* operation of two relational tables is formed by appending rows from one table to those of a second table to produce a third
xxiv. The *difference* of two relational tables is a third that contains those rows that occur in the first table but not in the second
xxv. The intersection of two relational tables is a third table that contains common rows.
xxvi. The product of two relational tables, also called the Cartesian product, is the concatenation of every row in one table with every row in the second.
xxvii. The *project* operator retrieves a *subset of columns* from a table, removing duplicate rows from the result.
xxviii. A *join* operation combines the product, selection, and, possibly, projection.

6.0 Tutor Marked Assignment

2. Briefly explain the following terms

i. Union
ii. Intersection
iii. Difference
iv. Cartesian product.
v. Selection
vi. Projection
vii. Join
viii. Division

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 2: Structured Query Language and Transaction Management

Unit 1: Structured Query Language (SQL)

1.0 Introduction

Structured Query Language (SQL) is a database computer language designed for managing data in relational database management systems (RDBMS). Its scope includes data query and update, schema creation and modification, and data access control.

SQL was developed at IBM by Andrew Richardson, Donald C. Messerly and Raymond F. Boyce in the early 1970s. This version, initially called **SEQUEL**, was designed to manipulate and retrieve data stored in IBM's original relational database product, System R.

SQL has two major parts:

a. Data Definition Language (DDL) Used to create (define) data structures such as tables, indexes, clusters
b. Data Manipulation Language (DML) is used to store, retrieve and update data from tables.

2.0 Objectives

By the end of this unit, you should be able to:

n. Know what relational algebra is all about

3.0 What Can SQL do?

a. SQL can execute queries against a database
b. SQL can retrieve data from a database
c. SQL can insert records in a database
d. SQL can update records in a database
e. SQL can delete records from a database
f. SQL can create new databases
g. SQL can create new tables in a database
h. SQL can create stored procedures in a database
i. SQL can create views in a database
j. SQL can set permissions on tables, procedures, and views

3.1 Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

Table 7.1: Persons Table
Source: http://www.w3schools.com/

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

3.2 SQL Data Types

Each implementation of SQL uses slightly different names for the data types.

3.2.1 Numeric Data Types

- Integers: `INTEGER`, `INT` or `SMALLINT`
- Real Numbers: `FLOAT`, `REAL`, `DOUBLE`, `PRECISION`
- Formatted Numbers: `DECIMAL(i,j)`, `NUMERIC(i,j)`

3.2.2 Character Strings

- Two main types: Fixed length and variable length.
- Fixed length of *n* characters: `CHAR(n)` or `CHARACTER(n)`
- Variable length up to size *n*: VARCHAR(n)

3.2.3 Date and Time

- Note: Implementations vary widely for these data types.
- `DATE`
  Has 10 positions in the format: `YYYY-MM-DD`
- `TIME`
  Has 8 positions in the format: `HH:MM:SS`
- `TIME(i)`
  Defines the TIME data type with an additional *i* positions for fractions of a second. For example:
  `HH:MM:SS:dd`
- Offset from UTZ. `+/- HH:MM`
- `TIMESTAMP`
- `INTERVAL`
  Used to specify some span of time measured in days or minutes, etc.
- Other ways of expressing dates:

  - Store as characters or integers with Year, Month Day:
    `19972011`
  - Store as Julian date:
    `1997283`
- Both MS Access and Oracle store date and time information together in a `DATE` data type.

3.2.4 Microsoft Access Data Types

Microsoft Access is one of the popular Relational Database Management System (RDBMS). The acceptable data types are as shown in table 7.2

Table 7.2: Microsoft Access Data Types
Source: http://www.w3schools.com/

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

3.2.5 MySQL Data Types

MySQL is another powerful RDBMS in use today. In MySQL there are three main data types: text, number, and Date/Time types (see figure 7.3 for more detail).

Table 7.3: Microsoft Access Data Types
Source: http://www.w3schools.com/

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis

MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date types:

Data type	**Description**
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format.

	Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD.

3.2.6 SQL Server Data Types

Table 7.4 lists some of the available Data Types in Microsoft SQL Sever.

Table 7.4: Microsoft Access Data Types
Source: **http://www.w3schools.com/**

Character strings:

Data type	Description	Storage
char(n)	Fixed-length character string. Maximum 8,000 characters	n
varchar(n)	Variable-length character string. Maximum 8,000 characters	
varchar(max)	Variable-length character string. Maximum 1,073,741,824 characters	
text	Variable-length character string. Maximum 2GB of text data	

Unicode strings:

Data type	Description	Storage
nchar(n)	Fixed-length Unicode data. Maximum 4,000 characters	
nvarchar(n)	Variable-length Unicode data. Maximum 4,000 characters	
nvarchar(max)	Variable-length Unicode data. Maximum 536,870,912 characters	
ntext	Variable-length Unicode data. Maximum 2GB of text data	

Binary types:

Data type	Description	Storage
bit	Allows 0, 1, or NULL	
binary(n)	Fixed-length binary data. Maximum 8,000 bytes	
varbinary(n)	Variable-length binary data. Maximum 8,000 bytes	
varbinary(max)	Variable-length binary data. Maximum 2GB	
image	Variable-length binary data. Maximum 2GB	

Number types:

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Date types:

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Other data types:

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

3.3 Data Definition Language (DDL)

The Data Definition Language (DDL) is used to create and destroy databases and database objects. Let us take a look at the structure and usage of basic DDL commands:

3.3.1 The CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a database. SQL CREATE DATABASE Syntax is:

```
CREATE DATABASE database_name
```

Example: Let us create a database called "my_db". We use the following CREATE DATABASE statement:

```
CREATE DATABASE my_db
```

This statement creates an empty database named "my_db" on your DBMS. After creating the database, your next step is to create tables that will contain data

3.3.2 The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database. SQL CREATE TABLE Syntax is:

```
CREATE TABLE table_name
(
column_name1  data_type,
column_name2  data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. See tables 7.2 to 7.4 for a complete reference of all the data types available in MS Access, MySQL, and SQL Server.

Example: Let us create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City. We use the following CREATE TABLE statement:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

3.3.3 SQL Constraints

Constraints are used to limit the type of data that can go into a table. Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

In this section, we will focus on the following constraints:

a. NOT NULL
b. UNIQUE
c. PRIMARY KEY
d. FOREIGN KEY

3.3.4 NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

3.3.5 SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Example1: The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

Example 2: To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

ALTER TABLE Persons
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)

Example 3: To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE Persons
DROP CONSTRAINT uc_PersonID

3.3.6 PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only one primary key.

Example 1:The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

Example 2: To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL:

ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)

3.3.7 SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let us illustrate the foreign key with an example. Look at table 7.1 above and table 7.5:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Table 7.5: Orders Table
Source: http://www.w3schools.com/

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy link between tables.

The FOREIGN KEY constraint also prevents that invalid data is inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Example 1:The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

Example 2: To create a FOREIGN KEY constraint on the "P_Id" column when the "Orders" table is already created, use the following SQL:

```
ALTER TABLE Orders
ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

3.3.7 USE

The USE command allows you to specify the database you wish to work with within your

DBMS.

USE employees

3.3.8 ALTER

Once you have created a table within a database, you may wish to modify the definition of it. The ALTER command allows you to make changes to the structure of a table without deleting and recreating it. Take a look at the following command:

```
ALTER TABLE personal_info
ADD salary money null
```

This example adds a new attribute to the personal_info table -- an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

3.3.9 DROP

The final command of the Data Definition Language, DROP, allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

```
DROP TABLE personal_info
```

Similarly, the command below would be used to remove the entire employees database:

```
DROP DATABASE employees
```

Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

Activity A

1a. what do you understand by Data Definition Language? Then list some of the available DDL commands

b. Write briefly on the following commands:

i. Create
ii. Use
iii. Alter
iv. Drop

3.4 Data Manipulation Language (DML)

Data Manipulation Language (DML) is used to manipulate (select, insert, update, delete) data.

3.4.1 The SQL SELECT Statement

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set. The SQL SELECT syntax:

```
SELECT column_name(s) FROM table_name

SELECT * FROM table_name
```

3.4.2 The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values. The syntax is

```
SELECT DISTINCT column_name(s) FROM table_name
```

3.4.3 The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion. The syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

Example:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

Note: SQL uses single quotes around text values (most database systems will also accept double quotes). Although, numeric values should not be enclosed in quotes.

The operators allowed in the WHERE clause are:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than

>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

3.4.4 The AND & OR Operators

The AND operator displays a record if both the first condition and the second condition is true.

Example

SELECT * FROM Persons WHERE FirstName='Tove' AND LastName='Svendson'

This will select only the persons with the first name equal to "Tove" AND the Last name equal to "Syendson":

The OR operator displays a record if either the first condition or the second condition is true.

Example

SELECT * FROM Persons WHERE FirstName='Tove' OR FirstName='Ola'

This will select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":

3.4.5 The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column. It sorts the records in ascending order by default. If you want to sort the records in a descending order, you can use the DESC keyword. The syntax is:

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

3.4.6 The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table. The syntax is:

```
INSERT INTO table_name VALUES (value1, value2, value3,...)

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

3.4.7 The UPDATE Statement

The UPDATE statement is used to update existing records in a table. The SQL UPDATE syntax is:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

3.4.8 The DELETE Statement

The DELETE statement is used to delete rows in a table. The SQL DELETE Syntax is:

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Activity B

Use the customer table shown in table 7.6 to answer the following SQL statements, displaying the resulting record sets

i. SELECT * FROM customers
ii. SELECT CompanyName, ContactName FROM customers
iii. SELECT * FROM customers WHERE companyname LIKE 'a%'
iv. SELECT CompanyName, ContactName FROM customers WHERE CompanyName > 'a'
v. SELECT CompanyName, ContactName FROM customers WHERE CompanyName > 'g' AND ContactName > 'g'

Table 7.6: Customers table
Source: Microsoft Northwind database sample

CompanyName	ContactName	Address	City
Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.
Ernst Handel	Roland Mendel	Kirchgasse 6	Graz

FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid
Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona
Island Trading	Helen Bennett	Garden House Crowther Way	Cowes
Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg
Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver
Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo

3.4.9 The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column. The SQL LIKE Syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

Example: If we want to select the persons living in a city that starts with "s" from the table 7.6; We use the following SELECT statement:

SELECT * FROM Persons WHERE City LIKE 's%'

The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

3.4.10 The IN Operator

The IN operator allows you to specify multiple values in a WHERE clause. The syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

Example: if we want to select the persons with a last name equal to "Hansen" or "Pettersen" from the Persons table; We use the following SELECT statement:
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')

3.4.11 The BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates. The SQL BETWEEN Syntax is:

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Example: If we want to select the persons with a last name alphabetically between "Hansen" and "Pettersen" from the persons table. We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

3.4.12 SQL JOIN

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys.

A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Look at the "Persons" table shown in table 7.1:

Persons table

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Note that the "P_Id" column is the primary key in the "Persons" table. This means that **no** two rows can have the same P_Id. The P_Id distinguishes two persons even if they have the same name.

Next, we have the "Orders" table shown in table 7.7:

Table 7.7: Orders Table
Source: **http://www.w3schools.com/**

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1

4	24562	1
5	34764	15

Note that the "O_Id" column is the primary key in the "Orders" table and that the "P_Id" column refers to the persons in the "Persons" table without using their names.

Notice that the relationship between the two tables above is the "P_Id" column.

The different types of Joins are listed below

- **JOIN**: Return rows when there is at least one match in both tables
- **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
- **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table

a. SQL INNER JOIN Keyword

The INNER JOIN keyword return rows when there is at least one match in both tables. The SQL INNER JOIN Syntax is:

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Example:

Using tables 7.1 and 7.7 above; if we want to list all the persons with any orders. We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	**FirstName**	**OrderNo**
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

b. SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2). The SQL LEFT JOIN Syntax is:

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

In some databases LEFT JOIN is called LEFT OUTER JOIN.

Example: If we ant to list all the persons and their orders - if any, from the tables 7.3 and 7.4 above. We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

c. SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1). SQL RIGHT JOIN Syntax is:

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Example: Let us list all the orders with containing persons - if any, from the tables 7.3 and 7.4 above. We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

d. The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order. SQL UNION Syntax:

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

Example: Let us consider tables 7.8:

Table 7.8: Employees table
Source: http://www.w3schools.com/

(a) Employees_Norway:

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

(b) Employees_USA:

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

(c) UNION of tables 7.8 a and b:

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

If we want to list **all the different** employees in Norway and USA; we use the following SELECT statement:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

The result-set will look like this table 7.8c

Activity C

1. Using tables 7.1 (Person's table) and 7.7 (Order's table), Demonstrate how to execute the following commands:

   i. Join
   ii. Left Join
   iii. Right Join
   iv. Full Join

4.0 Conclusion

When we wish to extract information from a database, we communicate with the Database Management System (DBMS) using a query language called SQL. SQL is the most frequently used programming language in the world, in the sense that every day, more SQL programs are written, compiled and executed than programs in any other computer programming language. SQL is used with *relational* database systems. In a relational database, all of the data is stored in tables.

5.0 Summary

In this unit, we have learnt:

xxix. **Structured Query Language** (SQL) is a database computer language designed for managing data in relational database management systems (RDBMS).
xxx. SQL has two major parts: Data Definition Language and Data Manipulation Language.
xxxi. Data Definition Language (DDL) Used to create (define) data structures such as tables, indexes, clusters
xxxii. Some of the available DDL commands are: Create, Use, Alter, and Drop
xxxiii. SQL Constraints are used to limit the type of data that can go into a table. The following constraint types were considered: Not Null, Unique, Primary Key, Foreign Key
xxxiv. Data Manipulation Language (DML) is used to manipulate (select, insert, update, delete) data in a Table.
xxxv. The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.
xxxvi. **JOIN**: Return rows when there is at least one match in both tables
xxxvii. **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
xxxviii. **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table.
xxxix. The UNION operator is used to combine the result-set of two or more SELECT statements.

6.0 Tutor Marked Assignment

PFNO	NAMES	STATUS	HIREDATESALARY		COMM	DEPTNO
1	AJAYI	CLERK	17-Dec-80	800		10
2	CHIM	SALESMAN	20-Feb-81	1600	300	40
3	JOHN	MANAGER	2-Apr-81	1250		40
4	WILL	SALESMAN	28-Sep-81	1250	300	30
5	KUDI	MANAGER	1-May-81	2975		30
6	TOLA	MANAGER	9-Jun-81	2850		20
7	ABDUL	ANALYST	27-Jun-90	3000		20
8	JAKE	PRESIDENT	3-Dec-81	5000		10
9	CLERK	SALESMAN	31-Jul-90	1234	500	40
10	SHEU	CLERK	3-Dec-81	1100		40
11	CHIDI	CLERK	3-Dec-81	950		20
12	HENRY	ANALYST	23-Jan-82	3000		20
13	IDIA	CLERK	23-Jan-82	1200		30
14	KUTI	SALESMAN	23-Jan-82	1600	600	20
15	BELLO	CLERK	23-Jan-82	1250		10

Employees Table

DEPTNO	DNAME	LOCATION
10	ACCOUNTING	IBADAN
20	RESEARCH	LAGOS
30	SALES	MINNA
40	OPERATION	KADUNA

Department Table

Use the above tables to answer the following questions:

Write an SQL statement that:

i. List all clerks who work in department 20
ii. List the names of all managers and analysts
iii. List the names of analysts who are not working in department 20
iv. List Employes with salary greater than 2500
v. `Which status are paid less than 2000 but more than 1000`
vi. `What is the total renumaration of sales people`
vii. List all employees in deparment 10, order by their salary
viii. Which employees work in Lagos
ix. Where does KUDI work?
x. Which postion are paid higher than average salary

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 2: Structured Query Language and Transaction Management

Unit 1: SQL Functions

1.0 Introduction

A *function* is a special type of command word in the SQL command set. In effect, functions are one-word commands that return a single value. The value of a function can be determined by input parameters, as with a function that averages a list of database values. But many functions do not use any type of input parameter, such as the function that returns the current system time, *CURRENT_TIME*.

The SQL supports a number of useful functions. This unit covers those functions, providing detailed descriptions and examples.

2.0 Objectives

By the end of this unit, you should be able to:

o. Perform arithmetic operations such as: finding average of column, finding sum of a column, finding the number of records in a table; finding the minimum and maximum values in a column.
p. Convert a field to upper or lower case
q. Extract characters from a text field
r. Format how a column or field should be displayed.

3.1 SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column. In this section, we discuss the following SQL aggregate commands. By the end of this section, you will learn the basics of retrieving data from the database using SQL.

Useful aggregate functions are:

i. AVG() - Returns the average value
ii. COUNT() - Returns the number of rows
iii. FIRST() - Returns the first value
iv. LAST() - Returns the last value
v. MAX() - Returns the largest value
vi. MIN() - Returns the smallest value
vii. SUM() - Returns the sum

3.1.1 The AVG Function

The AVG function returns the average value of a numeric column. AVG Syntax is:

```
SELECT AVG(column_name) FROM table_name
```

Example: Let us consider the following “Orders” table:

OrderId	OrderDate	Price	Customername
11	2008/11/12	1000	Henry Bank
21	2008/10/23	1600	Niyi Alade
31	2008/09/02	700	Henry Bank
41	2008/09/03	300	Henry Bank
51	2008/08/30	2000	James Adeola
61	2008/10/04	100	Niyi Alade

Question: Let us find the average value of the Price column.

Answer: We use the following SQL statement:

SELECT AVG(Price) AS AveragePrice FROM Orders

The result-set will look like this:

AveragePrice
950

We may decide to find the customers that have order Price value higher than the average Price value.

We use the following SQL statement:

SELECT Customername FROM Orders
WHERE Price>(SELECT AVG(Price) FROM Orders)

The result-set will look like this:

Customername
Henry Bank
Niyi Alade
James Adeola

3.1.2 The COUNT function

The COUNT function returns the number of rows that matches specified criteria. Note that null values will not be counted. In the section, we shall consider the following:

- SQL COUNT(column_name) Syntax
- SQL COUNT(*) Syntax
- SQL COUNT(DISTINCT column_name) Syntax

a. SQL COUNT(column_name) Syntax

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

Example: Let us consider our order table in section 3.11 again.

Now we want to count the number of orders from "Customer Niyi Alade".

We use the following SQL statement:

```
SELECT COUNT(Customer) AS NiyiAlade FROM Orders
WHERE Customer='Niyi Alade'
```

The result of the SQL statement above will be 2, because the customer Niyi Alade has made 2 orders in total:

NiyiAlade
2

b. SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

Example: Let us consider our order table again. Now we want to find the number of records in the order table.

We use the following SQL statement:

```
SELECT COUNT(*) AS NumberOfOrders FROM Orders
```

The result-set will look like this

NumberOfOrders
6

This is the total number of rows in the order table.

c. SQL COUNT(DISTINCT column_name) Syntax

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

SELECT COUNT(DISTINCT column_name) FROM table_name

Example: Now we want to count the number of unique customers in the "Orders" table.

We use the following SQL statement:

SELECT COUNT(DISTINCT Customer) AS TotalCustomers FROM Orders

The result-set will look like this:

TotalCustomers
3

3.1.3 The FIRST Function

The FIRST function returns the first value of the selected column. The SQL Syntax is:

SELECT FIRST(column_name) FROM table_name

Example: We will still make use of our orders table in section 3.11

Now we want to find the first value of the "Price" column.

We use the following SQL statement:

SELECT FIRST(Price) AS FirstPrice FROM Orders

The result-set will look like this:

FirstOrderPrice
1000

3.1.4 The LAST Function

The LAST function returns the last value of the selected column. The syntax is:

SELECT LAST(column_name) FROM table_name

Example: We have the "Orders table in section 3.11

Now we want to find the last value of the Price column.

We will make use of the following SQL statement:

SELECT LAST(Price) AS LastOrderPrice FROM Orders

The result-set will look like this:

LastOrderPrice
100

3.1.5 The MAX Function

The MAX function returns the largest value of the selected column. The SQL MAX Syntax is:

SELECT MAX(column_name) FROM table_name

Example: Let us consider Orders table again:

This time around we want to find the largest value of the Price column.

We shall make use of the following SQL statement:

SELECT MAX(Price) AS LargestPrice FROM Orders

The result-set will look like this:

LargestPrice
2000

3.1.6 The MIN Function

The MIN function returns the smallest value of the selected column. The SQL MIN Syntax is as follows:

SELECT MIN(column_name) FROM table_name

Example from our Orders table: let us find the smallest value of the Price column.

We use the following SQL statement:

SELECT MIN(Price) AS SmallestPrice FROM Orders

The result-set will look like this:

SmallestPrice
100

3.1.7 SUM Function

The SUM function is used to calculate the total for a column. The syntax is,

```
SELECT SUM("column_name") FROM "table_name"
```

Example from Orders table: we want to find the sum of all Price field.

We use the following SQL statement:

SELECT SUM(Price) AS OrderTotal FROM Orders

The result-set will look like this:

OrderTotal
5700

3.1.8 The GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns. The syntax is:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

Example: let us consider the Orders table again:

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following SQL statement:

SELECT Customername, SUM(Price) FROM Orders
GROUP BY Customername

The result-set will look like this:

Customername	SUM(Price)
Henry Bank	2000
Niyi Alade	1700
James Adeola	2000

Let us see what will happen if we omit the GROUP BY statement:

SELECT Customername, SUM(Price) FROM Orders

The result-set will look like this:

Customername	SUM(Price)
Henry Bank	5700
Niyi Alade	5700
Henry Bank	5700
Henry Bank	5700
James Adeola	5700
Niyi Alade	5700

The result-set above is not what we wanted.

3.1.9 The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The syntax is:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

Example: Now we want to find if any of the customers have a total order of less than 2000.

We use the following SQL statement:

```
SELECT Customername, SUM(Price) FROM Orders
GROUP BY Customer
HAVING SUM(Price)<2000
```

The result-set will look like this:

Customername	SUM(Price)

Niyi Alade	1700

Now we want to find if the customers "Henry Bank" or "James Adeola" have a total order of more than 1500.

We add an ordinary WHERE clause to the SQL statement:

```
SELECT Customername, SUM(Price) FROM Orders
WHERE Customername='Henry Bank' OR Customername='James Adeola'
GROUP BY Customername
HAVING SUM(Price)>1500
```

The result-set will look like this:

Customername	**SUM(Price)**
Henry Bank	2000
James Adeola	2000

Activity A

1. Write out the SQL syntax for the following functions
   i. AVG()
   ii. COUNT()
   iii. FIRST()
   iv. LAST()
   v. MAX()
   vi. MIN()
   vii. SUM()

3.2 SQL Scalar functions

SQL scalar functions return a single value, based on the input value. In this section, we discuss the following SQL scalar commands. By the end of this section, you will learn the basics of manipulating data from the database using SQL.

Some useful scalar functions are:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

We shall make use of the following Persons table throughout this section

PersonId	Surname	Firstname	Address	City
1	Henry	Bank	15 Allen Avenue	Lagos
2	Ebuka	Tunji	23 Wuse Zone 4	Abuja
3	Peter	Kasim	78 Baba street	Kaduna

3.2.1 The UCASE Function

The UCASE function is used to convert the value of a column to uppercase. The syntax is

SELECT UCASE (column_name) FROM table_name

Example: We have a Persons table in section 3.2, now we want to select the content of the Surname and FirstName columns, and convert the Surname column to uppercase.

We use the following SELECT statement:

SELECT UCASE(Surname) as Surname, FirstName FROM Persons

The result-set will look like this:

Surname	FirstName
HENRY	Bank
EBUKA	Tunji
PETER	Kasim

3.2.2 The LCASE Function

The LCASE() function converts the value of a column to lowercase. The syntax is:

SELECT LCASE(column_name) FROM table_name

Example: Let us select the content of the Surname and FirstName columns from our Persons table, and convert the Surname column to lowercase.

We use the following SELECT statement:

SELECT LCASE(Surname) as Surname, FirstName FROM Persons

The result-set will look like this:

Surname	FirstName

henry	Bank
ebuka	Tunji
peter	Kasim

3.2.3 The MID Function

The MID function is used to extract characters from a text column. The syntax is:

SELECT MID(column_name,start[,length]) FROM table_name

Parameters:	**Description**
column_name:	Required. The column to extract characters from
start:	Required. Specifies the starting position (starts at 1)
length:	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

Example: Let us extract the first four characters of the "City" column from Persons table.

We use the following SELECT statement:

SELECT MID(City,1,4) as City FROM Persons

The result-set will look like this:

City
Lago
Abuj
Kadu

3.2.4 The LEN Function

The LEN function returns the length of the value in a text column. The syntax is:

SELECT LEN(column_name) FROM table_name

Example: Let us select the length of the values in the Address column of Persons table.

We use the following SELECT statement:

SELECT LEN(Address) as LengthOfAddress FROM Persons

The result-set will look like this:

LengthOfAddress
15
14
14

3.2.5 The ROUND() Function

The ROUND function is used to round a numeric field to the number of decimals specified. The syntax is:

SELECT ROUND(column_name,decimals) FROM table_name

Parameter	Description
column_name	Required. The field to round.
Decimals	Required. Specifies the number of decimals to be returned.

Example: Let us consider the Products table below:

ProductID	ProductName	Unit	UnitPrice
11	Sugar	1000 g	10.45
12	Salt	1000 g	32.56
13	Palm Oil	1000 g	15.67

Now we want to display the product name and the price rounded to the nearest integer.

We use the following SELECT statement:

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products

The result-set will look like this:

ProductName	UnitPrice
Sugar	10
Salt	33
Palm Oil	16

3.2.6 The NOW Function

The NOW function returns the current system date and time. The syntax is:

SELECT NOW() FROM table_name

Example: Let us consider the product table again. Now we want to display the products and prices per today's date.

We use the following SELECT statement:

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

The result-set will look like this:

ProductName	UnitPrice	PerDate
Sugar	10.45	8/18/2009 10:35:02 AM
Salt	32.56	8/18/2009 10:35:02 AM
Palm Oil	15.67	8/18/2009 10:35:02 AM

3.2.7 The FORMAT Function

The FORMAT function is used to format how a field is to be displayed. The syntax is:

SELECT FORMAT(column_name,format) FROM table_name

Parameter	Description
column_name	Required. The field to be formatted.
Format	Required. Specifies the format.

Example: Let us make use of the products table here. Now we want to display the products and prices per today's date (with today's date displayed in the following format "YYYY-MM-DD").

We use the following SELECT statement:

SELECT ProductName, UnitPrice, FORMAT(Now(),'YYYY-MM-DD') as PerDate FROM Products

The result-set will look like this:

ProductName	UnitPrice	PerDate
Sugar	10.45	2009/8/18
Salt	32.56	2009/8/18
Palm Oil	15.67	2009/8/18

Activity B

1. Write out the SQL syntax for the following functions:

i. UCASE
ii. LEN
iii. MID
iv. LCASE
v. ROUND
vi. NOW
vii. FORMAT

4.0 Conclusion

SQL has many built-in functions for performing calculations on data. These functions were categorized into: SQL Aggregate functions and SQL Scalar functions. The aggregate functions operate against a collection of values, but return a single, summarizing value. Scalar functions Operate against a single value, and return a single value based on the input value. Some scalar functions, *CURRENT_TIME* for example, do not require any arguments.

5.0 Summary

In this unit, we have learnt:

xl. The basics of retrieving data from the database using SQL.
xli. AVG function is to return the average value of a column in a database table.
xlii. COUNT function returns the number of rows in a database table.
xliii. FIRST function returns the first value in a database table.
xliv. LAST function returns the last value in a database table.
xlv. MAX function returns the largest value
xlvi. MIN function returns the smallest value
xlvii. SUM function returns the sum
xlviii. UCASE function converts a field to upper case
xlix. LCASE converts a field to lower case
l. MID function extract characters from a text field
li. LEN function returns the length of a text field
lii. ROUND function rounds a numeric field to the number of decimals specified
liii. NOW function returns the current system date and time
liv. FORMAT function formats how a field is to be displayed

6.0 Tutor Marked Assignment

PFNO	NAMES	STATUS	HIREDATESALARY	SALARY	COMM	DEPTNO
1	AJAYI	CLERK	17-Dec-80	800		10
2	CHIM	SALESMAN	20-Feb-81	1600	300	40

3	JOHN	MANAGER	2-Apr-81	1250		40
4	WILL	SALESMAN	28-Sep-81	1250	300	30
5	KUDI	MANAGER	1-May-81	2975		30
6	TOLA	MANAGER	9-Jun-81	2850		20
7	ABDUL	ANALYST	27-Jun-90	3000		20
8	JAKE	PRESIDENT	3-Dec-81	5000		10
9	CLERK	SALESMAN	31-Jul-90	1234	500	40
10	SHEU	CLERK	3-Dec-81	1100		40
11	CHIDI	CLERK	3-Dec-81	950		20
12	HENRY	ANALYST	23-Jan-82	3000		20
13	IDIA	CLERK	23-Jan-82	1200		30
14	KUTI	SALESMAN	23-Jan-82	1600	600	20
15	BELLO	CLERK	23-Jan-82	1250		10

Employees Table

From the above tables, write the SQL statement that:

xi. Calculate the employees salary
xii. Find the number of employees
xiii. Find the highest salary
xiv. Find the total sum of salary paid
xv. Find the total salary for each status
xvi. Which positions are paid higher than average salary?

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 2: Structured Query Language and Transaction Management

Unit 3: Transactions and Concurrency Management

1.0 Introduction

In a database management system (DBMS), a transaction consists of one or more data-manipulation statements and queries, each of which is reading and/or writing information into the database. A transaction is usually issued to the DBMS in SQL language wrapped in a transaction, using a pattern similar to the following:

- Begin the transaction
- Execute several data manipulations and queries
- If no errors occur then commit the transaction and end it
- If errors occur then rollback the transaction and end it

If there is no error during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and store the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state.

In this unit, we shall make use of use the pubs database found in Microsoft SQL Server DBMS if the need arises.

2.0 Objectives

By the end of this unit, you should be able to know:

a. What a database transaction is and what its properties are
b. How database transactions are managed
c. What concurrency control is and what role it plays in maintaining the database integrity
d. What locking methods are and how they work
e. How database recovery management is used to maintain database integrity

3.0 MultiUser Databases

Multi-user database access is one of the DBMS properties that motivate transactions. In a Multi-user database more than one user processes the database at the same time. Several issues arise:

- How can we prevent users from interfering with each other's work?
- How can we safely process transactions on the database without corrupting or losing data?
- If there is a problem (e.g., power failure or system crash), how can we recover without loosing all of our data?

3.1 What is a Transaction?

A transaction is a logical unit of database processing, which include one or more database operations, such as insertion, deletion, modification, or retrieval operations.

Transaction processing systems are systems with large databases and hundreds of concurrent users.

A transaction must be entirely completed or aborted, no intermediate states are acceptable.
A consistent database state is one in which all data integrity constraints are satisfied.

A transaction can also be defined as a sequence of operations performed as a single logical unit of work. A transaction has four key properties that are abbreviated ACID. ACID is an acronym for Atomic, Consistent, Isolated, and Durability.

a. Atomic means that all the work in the transaction is treated as a single unit. Either it is all performed or none of it is.
b. Consistent means that a completed transaction leaves the database in a consistent internal state.
c. Isolations mean that the transaction sees the database in a consistent state. This transaction operates on a consistent view of the data. If two transactions try to update the same table, one will go first and then the other will follow.
d. Durability means that the results of the transaction are permanently stored in the system.

The simplest transaction in SQL Server is a single data modification statement. The following is a transaction even though it does not do much.

```
UPDATE  authors
SET     au_fname = 'John'
WHERE   au_id = '172-32-1176'
```

SQL Server first writes to the log file what it is going to do. Then it does the actual update statement and finally it writes to the log that it completed the update statement. The writes to the log file are written directly to disk but the update itself is probably done to a copy of the data that resides in memory. At some future point that database will be written to disk. If the server fails after a transaction has been committed and written to the log, SQL Server will use the transaction log to "roll forward" that transaction when it starts up next.

3.1.1 Transaction Examples

Consider two users, each executing similar transactions:

Example #1:

User A	User B
Read Salary for emp 101	Read Salary for emp 101
Multiply salary by 1.03	Multiply salary by 1.04
Write Salary for emp 101	Write Salary for emp 101

Example #2:

User A	User B
Read inventory for Prod 200	Read inventory for Prod 200
Decrement inventory by 5	Decrement inventory by 7
Write inventory for Prod 200	Write inventory for Prod 200

First, what should the values for salary (in the first example) really be?

The DBMS must find a way to execute these two transactions concurrently and ensure the result is what the users (and designers) intended.

These two are examples of the Lost Update or Concurrent Update problem. Some changes to the database can be overwritten.

Consider how the operations for user's A and B might be interleaved as in example #2. Assume there are 10 units in inventory for Prod 200:

Read inventory for Prod 200	for user A
Read inventory for Prod 200	for user B
Decrement inventory by 5	for user A
Decrement inventory by 7	for user B
Write inventory for Prod 200	for user A
Write inventory for Prod 200	for user B

Or something similar like:

Read inventory for Prod 200	for user A
Decrement inventory by 5	for user A
Write inventory for Prod 200	for user A
Read inventory for Prod 200	for user B
Decrement inventory by 7	for user B
Write inventory for Prod 200	for user B

In the first case, the incorrect amount is written to the database. This is called the **Lost Update** problem because we lost the update from User A - it was overwritten by user B.

The second example works because we let user A write the new value of Prod 200 before user B can read it. Thus User B's decrement operation will fail.

Here is another example. User's A and B share a bank account. Assume an initial balance of $200.

User A reads the balance
User A deducts $100 from the balance
User B reads the balance
User A writes the new balance of $100
User B deducts $100 from the balance
User B writes the new balance of $100

The reason we get the wrong final result (remaining balance of $100) is because transaction B was allowed to read stale data. This is called **the inconsistent read** problem.

Suppose, instead of interleaving (mixing) the operations of the two transactions, we execute one after the other (note it makes no difference which order: A then B, or B then A)

User A reads the balance
User A deducts $100 from the balance
User A writes the new balance of $100
User B reads the balance (which is now $100)
User B deducts $100 from the balance
User B writes the new balance of $0

If we insist only one transaction can execute at a time, in serial order, then performance will be quite poor.

3.1.2 Multi-Statement Transactions

To make transactions a little more useful we really need to put two or more statements in them. These are called **Explicit Transactions**. For example,

```
BEGIN TRAN

UPDATE   authors
SET      au_fname = 'John'
WHERE    au_id = '172-32-1176'

UPDATE   authors
SET      au_fname = 'Marg'
WHERE    au_id = '213-46-8915'

COMMIT TRAN
```

Note that we have a BEGIN TRAN at the beginning and a COMMIT TRAN at the end. These statements start and complete a transaction. Everything inside these statements is considered a logical unit of work. If the system fails after the first update, neither update statement will be applied when SQL Server is restarted. The log file will contain a BEGIN TRAN but no corresponding COMMIT TRAN.

3.2 Transaction Management with SQL

Transaction support:

a. Commit
b. Rollback

User initiated transaction is executed in sequence until:

a. Commit statement is reached
b. Rollback statement is reached
c. End of a program is reached
d. Program reaches abnormal termination which leads to rollback

3.2.1 Rolling Back

You can also roll back a transaction if it does not do what you want. Consider the following transaction:

```
BEGIN TRAN

UPDATE   authors
SET      au_fname = 'John'
WHERE    au_id = '172-32-1176'

UPDATE   authors
SET      au_fname = 'JohnY'
WHERE    city = 'Lawrence'

IF @@ROWCOUNT = 5
  COMMIT TRAN
ELSE
  ROLLBACK TRAN
```

Suppose that for whatever reason, the second update statement should update exactly five rows. If @@ROWCOUNT, which hold the number of rows affected by each statement, is five then the transaction commits otherwise it rolls back. The ROLLBACK TRAN statement undoes all the work since the matching BEGIN TRAN statement. It will not perform either update statement. Note that Query Analyzer will show you messages indicating that rows were updated but you can query the database to verify that no actual data modifications took place.

3.2.2 Transaction Log

The following are the functions of transaction log:

a. It tracks all transactions that update the database
b. It may also be used by rollback command
c. It may be used to recover from system failure
d. It log record for beginning of a transaction
e. It also log each SQL statement which include:
    i. Operation type (retrieve, update, insert, delete)
    ii. Names of objects
    iii. Before and after values for updated fields
    iv. Pointers to previous and next entries
f. It log commit statement

3.2.3 Stored Procedures

A **stored procedure** is a subroutine available to applications accessing a relational database system. Stored procedures are actually stored in the database data dictionary.

Typical uses for stored procedures include data validation (integrated into the database) or access control mechanisms. Furthermore, stored procedures are used to consolidate and centralize logic that was originally implemented in applications. Large or complex processing that might require the execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

Hopefully most of transactions will occur in stored procedures. Let us look at the second example inside a stored procedure.

```
Create Proc TranTest1
AS
BEGIN TRAN

INSERT INTO [authors]
            ([au_id],
            [au_lname],
            [au_fname],
            [phone],
            [contract])

VALUES  ('172-32-1176',
            'Gates',
            'Bill',
            '800-BUY-MSFT',
            1)

UPDATE  authors
SET     au_fname = 'Johnzzz'
WHERE   au_id = '172-32-1176'

COMMIT TRAN
GO
```

The problem with this stored procedure is that transactions do not care if the statements run correctly or not. They only care if SQL Server failed in the middle. If you run this stored procedure, it will try to insert a duplicate entry into the authors database. You will get a primary key violation error message. The message will even tell you the statement has been terminated. But the transaction is still going. The UPDATE statement runs just fine and SQL Server then commits the transaction. The proper way to code this is:

```
Create Proc TranTest2
AS
BEGIN TRAN

INSERT INTO [authors]([au_id],
            [au_lname],
            [au_fname],
            [phone],
            [contract])
VALUES  ('172-32-1176',
            'Gates',
            'Bill',
            '800-BUY-MSFT',
            1)

IF @@ERROR <> 0
  BEGIN
            ROLLBACK TRAN
            return 10
  END

UPDATE  authors
SET     au_fname = 'Johnzzz'
WHERE   au_id = '172-32-1176'

IF @@ERROR <> 0
  BEGIN
            ROLLBACK TRAN
            return 11
  END

COMMIT TRAN
GO
```

You will notice that we check each statement for failure. If the statement failed (i.e. @@ERROR <> 0) then we rollback the work performed so far and use the RETURN statement to exit the stored procedure. It is very important to note that if we do not check for errors after each statement we may commit a transaction improperly.

Activity A

1. What is Transaction?
   What are the properties of a transaction?

2. Explain the following transaction problems with examples

i. Concurrent Update problem
ii. Inconsistent read problem.

3.3 Concurrency Control and Locking

We need the ability to control how transactions are run in a multiuser database. Let us define some basic terms that are related to concurrency control:

a. **Concurrency Control** is a method for controlling or scheduling the operations in such a way that concurrent transactions can be executed. If we do concurrency control properly, then we can maximize transaction throughput while avoiding any chance.
    - Concurrency control ensure serializability of transactions in multiuser environment
    - It solve problems in multiuser environment which include:
        - Lost Updates
        - Uncommitted data
        - Inconsistent retrievals
b. **Transaction throughput**: The number of transactions we can perform in a given time period. Often reported as Transactions per second or TPS.
c. A group of two or more concurrent transactions are serializable if we can order their operations so that the final result is the same as if we had run them in serial order (one after another).
d. Consider transaction A, B, C and D. Each has 3 operations. If executing:
   A1, B1, A2, C1, C2, B2, A3, B3, C3
   has the same result as executing:
   A1, A2, A3, B1, B2, B3, C1, C2, C3
   Then the above schedule of transactions and operations is serialized.
e. We need a way to guarantee that our concurrent transactions can be serialized. Locking is one such means. **Locking** is done to data items in order to reserve them for future operations.
f. A lock is a logical flag set by a transaction to alert other transactions the data item is in use.

3.3.1 The Scheduler

Scheduler establishes order of concurrent transaction execution. It interleaves execution of database operations to ensure serializability. It based its actions on concurrency control algorithms which are Locking and Time stamping.

3.3.2 Characteristics of Locks

Locks may be applied to data items in two ways: Implicit and Explicit.

Implicit Locks are applied by the DBMS, while **Explicit Locks** are applied by application programs.

Locks may be applied to:

i. a single data item (value)
ii. an entire row of a table
iii. a page (memory segment) (many rows worth)
iv. an entire table
v. an entire database

This is referred to as the Lock granularity.

Locks may be of the following types depending on the requirements of the transaction:

i. An **Exclusive Lock** prevents any other transaction from reading or modifying the locked item.
ii. A **Shared Lock** allows another transaction to read an item but prevents another transaction from writing the item.

3.3.3 Two Phase Locking

The most commonly implemented locking mechanism is called Two Phased Locking or **2PL**. 2PL is a concurrency control mechanism that ensure serializability.

2PL has two phases: Growing and shrinking.

i. A transaction acquires locks on data items it will need to complete the transaction. This is called the **growing phase**.
ii. Once one lock is released, all no other lock may be acquired. This is called the **shrinking phase**.

Let us consider our previous examples in section 3.11, this time using exclusive lock:

User A places an exclusive lock on the balance
User A reads the balance
User A deducts $100 from the balance

User B attempts to place a lock on the balance
but fails because A already has an exclusive lock
User B is placed into a wait state
User A writes the new balance of $100
User A releases the exclusive lock on the balance

User B places an exclusive lock on the balance
User B reads the balance
User B deducts $100 from the balance
User B writes the new balance of $100

Here is a more involved illustration using exclusive and shared locks:

User A places a shared lock on item **raise_rate**
User A reads **raise_rate**
User A places an exclusive lock on item **Amy_salary**
User A reads **Amy_salary**

User B places a shared lock on item **raise_rate**
User B reads **raise_rate**

User A calculates a new salary as Amy_salary * (1+raise_rate)

User B places an exclusive lock on item **Bill_salary**
User B reads **Bill_salary**
User B calculates a new salary as Bill_salary * (1+raise_rate)
User B writes **Bill_salary**

User A writes **Amy_salary**
User A releases exclusive lock on **Amy_salary**

User B releases exclusive lock on **Bill_Salary**
User B releases shared lock on **raise_rate**

User A releases shared lock on **raise_rate**

Here is another example:

User A places a shared lock on **raise_rate**

User B attempts to place an exclusive lock on **raise_rate**
Placed into a wait state

User A places an exclusive lock on item **Amy_salary**
User A reads **raise_rate**
User A releases shared lock on **raise_rate**

User B places an exclusive lock on **raise_rate**

User A reads **Amy_salary**

User B reads **raise_rate**

User A calculates a new salary as Amy_salary * (1+raise_rate)

User B writes a new **raise_rate**
User B releases exclusive lock on **raise_rate**

User A writes **Amy_salary**
User A releases exclusive lock on **Amy_salary**

3.3.4 Deadlock

Deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a *software lock* or *soft lock*.

Deadlock occurs when two transactions wait for each other to unlock data.

Example of deadlock in database as follows:

Client applications using the database may require exclusive access to a table, and in order to gain exclusive access they ask for a *lock*. If one client application holds a lock on a table and attempts to obtain the lock on a second table that is already held by a second client application, this may lead to deadlock if the second application then attempts to obtain the lock that is held by the first application. (But this particular type of deadlock is easily prevented, e.g., by using an *all-or-none* resource allocation algorithm.)

Locking can cause problems, however, let us consider another example:

User A places an exclusive lock on item 1001
User B places an exclusive lock on item 2002
User A attempts to place an exclusive lock on item 2002
    User A placed into a wait state
User B attempts to place an exclusive lock on item 1001
    User B placed into a wait state

This is also called a **deadlock**. One transaction has locked some of the resources and is waiting for locks so it can complete. A second transaction has locked those needed items but is awaiting the release of locks the first transaction is holding so it can continue.

3.3.5 How Prevent Deadlock

Two main ways to deal with deadlock.

1. Prevent it in the first place by giving each transaction exclusive rights to acquire all locks needed before proceeding.
2. Allow the deadlock to occur, and then break it by aborting one of the transactions.

3.4 Database Recovery Management

There are many situations in which a transaction may not reach a commit or abort point.

a. If an operating system crashes it can terminate the DBMS processes
b. The DBMS can crash
c. Power failure i.e., The system might lose power
d. Disk or hardware failure.
e. Human error can result in deletion of critical data.

In any of these situations, data in the database may become inconsistent or lost.

Database Recovery is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).

Two approaches are discussed in this section: Reprocessing and Rollback/Rollforward.

3.4.1 Reprocessing

In a **Reprocessing** approach, the database is periodically backed up (a database save) and all transactions applied since the last save are recorded

If the system crashes, the latest database save is restored and all of the transactions are re-applied (by users) to bring the database back up to the point just before the crash.

3.4.2 Automated Recovery with Rollback / Rollforward

In this approach, we apply a similar technique: Make periodic saves of the database (time consuming operation). However, maintain a more intelligent log of the transactions that have been applied. This transaction log Includes before images and after images

- **Before Image**: A copy of the table record (or page) of data before it was changed by the transaction.
- **After Image**: A copy of the table record (or page) of data after it was changed by the transaction.
- **Rollback**: Undo any partially completed transactions (ones in progress when the crash occurred) by applying the before images to the database.
- **Rollforward**: Redo the transactions by applying the after images to the database. This is done for transactions that were committed before the crash.
- Recovery process uses both rollback and rollforward to restore the database.
- In the worst case, we would need to rollback to the last database save and then rollforward to the point just before the crash.
- **Checkpoints** can also be taken (less time consuming) in between database saves.
- The DBMS flushes all pending transactions and writes all data to disk and transaction log.

- Database can be recovered from the last checkpoint in much less time.

3.5 Database Backup

When secondary media (disk) fails, data may become unreadable. We typically rely on backing up the database to cheaper magnetic tape or other backup medium for a copy that can be restored. However, when a DBMS is running, it is not possible to backup its files as the resulting backup copy on tape may be inconsistent.

One solution: Shut down the DBMS (and thus all applications), do a full backup - copy everything on to tape. Then start up again.

Most modern DBMS allow for incremental backups.

- An **Incremental backup** will backup only those data changed or added since the last full backup. Sometimes called a delta backup.
- Follows something like:
    a. Weekend: Do a shutdown of the DBMS, and full backup of the database onto a fresh tape(s).
    b. Nightly: Do an incremental backup onto different tapes for each night of the week.

Activity B

1. Explain the terms in relation to database transactions management:
   i. Concurrency Control
   ii. Transaction Throughput
   iii. Serialization
   iv. Implicit Locking

2. What is deadlock? How can deadlock be prevented?

4.0 Conclusion

Database transaction management is a crucial issue in order to maintain data consistency.

5.0 Summary

In this unit, we have learnt:

lv. Transactions are set of read and write operations that must either execute to completion or not at all.
lvi. A transaction has four key properties that are abbreviated ACID. ACID is an acronym for Atomic, Consistent, Isolated, and Durability.
lvii. Concurrency Control is a method for controlling or scheduling the operations in such a way that concurrent transactions can be executed.

lviii. Transaction throughput is the number of transactions we can perform in a given time period.
lix. A group of two or more concurrent transactions are serializable if we can order their operations so that the final result is the same as if we had run them in serial order (one after another).
lx. Locking is one of the ways to guarantee that our concurrent transactions can be serialized.
lxi. Locks may be applied to data items in two ways: Implicit and Explicit
lxii. Deadlock refers to a specific condition when two or more transactions are each waiting for each other to release a lock.
lxiii. Database Recovery is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).
lxiv. An **Incremental backup** will backup only those data changed or added since the last full backup.

6.0 Tutor Marked Assignment

1. Explain the following SQL Transaction terms with an example:

   i. Lost Update problem
   ii. Dirty read
   iii. Non-repeatable read
   iv. Phantom read

2. Explain the following Database recovery terms:

   i. Before Image
   ii. After image
   iii. Rollback
   iv. Roll forward
   v. Checkpoints

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.
w3schools.com (2009). SQL Tutorial. Retrieved April 10th, 2009, from: http://www.w3schools.com/sql/

Module 2: Structured Query Language and Transaction Management

Unit 4: Database Security

1.0 Introduction

Database security refers to the protection of data against unauthorized access, alteration, or destruction. System needs to be aware of certain constraints that users must nit violate; those constraints must be specified in a suitable language and must be maintained in the system catalog; DBMS must monitor users to ensure that the constraints are enforced.

Security in a database involves mechanisms to protect the data and to ensure that it is not accessed, altered or deleted without proper authorization. Access to data should be restricted and should be protected from accidental destruction.

This unit provides an overview of database security and recovery. The need for database security, classification of database security and different type of database failures were discussed.

2.0 Objectives

By the end of this unit, you should be familiar with the following concepts:

s. Need for database security
t. Classification of database security
u. Database security at design and maintenance Level
v. Database security through access control

3.1 Database Security

The issues relating to database security will be discussed in this section.

3.1.1 Need for Database Security

The need for database security is given below:

a. In a multi-user database, multiple users try to access the data at the same time. In order to maintain the consistency of the data, database security is needed
b. For the data that are being accessed through the World Wide Web, there is need to protect the data against hackers.
c. The use of credit cards is becoming more popular, the money transaction has to be saved. More specialized software both to enter the system illegally to extract data and to analyze the information obtained are available. Hence, it is necessary to protect the data.

3.1.2 Approaches to Data Security

There are numerous aspects to the security problem, some of them are:

a. Legal, social, and ethical aspects
b. Physical controls

c. Policy questions
d. Operational problems
e. Hardware control
f. Operating system support
g. Issues that the specific concern of the database system itself

As database practitioners, we must provide a means of preventing unauthorized use of data in a database. Three areas are considered:

a. Access Control: Who should be allowed access to which databases? This is typically enforced using system level accounts with passwords.
b. Authorization: for the purposes of:
   Reading Data - Such as reading another employee's salary (using a SELECT statement for example).
   Writing Data - Such as changing a value in a database (using UPDATE or DELETE).
c. Statistical Information: Enforcing who should be allowed access to information derived from underlying databases.

3.1.3 Database Security Goals and Threats

Some of the goals of Database security are:

a. Confidentiality (Secrecy and Privacy – data are only accessible by authorized users
b. To ensure data integrity which means data can only be modified by authorized users
c. Availability – data are accessible to authorized users

Some of the threats of database security are:

a. Improper release of information caused by reading of data through intentional or accidental access by unauthorized users
b. Improper modification of data
c. Action could prevent users from accessing data for which they are authorized

3.1.4 Security Threat Classification

Security threat can be classified into accidental and intentional, according to the way they occur.

The accidental threats include human errors, software errors, and natural or accidental disasters:

a. Human errors include giving incorrect input and incorrect use of applications

b. Software errors include incorrect application of security policies, denial of access to authorized users.
c. Natural or accidental threat includes damage of hardware or software

The intentional threat includes authorized users who abuse their privileges and authority, hostile agents like unauthorized users executing improper reading or writing of data.

3.1.5 Classification of Database Security

Database security can be classified into physical and logical security.

a. Physical Security: this refers to the security of hardware associated with the system and the protection of site where the computer resides. Natural events such as fire, floods, and earthquakes can be considered as part of the physical threats. It is advisable to have backup copies of databases in the face of massive disasters.
b. Logical Security: This refers to the security measures residing in the operating system or the DBMS to handle threats to the data.

3.1.6 Database Security at Design Level

It is necessary to take care of security at the database design stage. The following guidelines should be put into consideration at the design stage:

a. The database should be simple and easier to use.
b. Database must be normalized
c. You should decide the privilege for each group of users.
d. Create unique view for each user or group of users.

3.1.7 Database Security at Maintenance Level

The security issues with respect to maintenance are as follows:

a. Operating system availability
b. Confidentiality and Accountability through Authorization rules
c. Encryption
d. Authentication Scheme

a. Operating system Availability: The operating system should verify that users and application programs attempting to access the system are authorized.

b. Confidentiality and Accountability: Accountability means that the system does not allow illegal entry. Accountability is related to both prevention and detection of illegal actions. Accountability is assured by monitoring the authentication and authorization of users.

Authorization rules are controls incorporated in the data management system that restrict access to data and also restrict the actions that people may take when they access data

Authentication may be carried out by the operating system or the relational database management system. Users are giving individual account or username and password.

c. Encryption: This is the coding of data so that they cannot be read and understood easily. Some DBMS products include encryption routines that automatically encode sensitive data when they are stored. They also provide complementary routines for decoding the data.

Activity A

1. What are the goals of database security?
2. What do you understand by database security threat?

3.1.8 Database Security System

The person responsible for security of the database is database administrator (DBA). The database administrator must consider variety of potential threats to the system. Database administrators create authorization rules that define who can access what parts of database for what operations. Enforcement of authorization rules involves authenticating the user and ensuring that the authorization rules are not violated by access request.

The database security system stores authorization rules and enforces them for each database access. When a group of users access the data in the database, then privileges may be assigned to the groups rather than individual users. In this section we shall consider authorization subsystem of database security system

3.1.9 Authorization Subsystem

Typically security for database authorization purposes is implemented in an *authorization subsystem* that monitors every transaction in the database. This is part of the DBMS

Authorization rules take into account a few main ideas:

i. **Subjects**: Individuals who perform some activity on the database. Might include specific people or a group of users.
ii. **Objects**: Database units that require authorization in order to manipulate. Database units might include an entire table, specific columns in a table, specific rows in a table, etc.
iii. **Actions**: Any action that might be performed on an object by a subject. For example: Read, Modify, Insert, Write, Delete, Grant (the ability to grant authorizations to others)

iv. **Constraint**: A more specific rule regarding an aspect of the object and action.

These elements are commonly combined into an *authorization table*

Subject	Object	Action	Constraint
Rasheed	Employee	Read	None
Rasheed	Employee	Insert	None
Rasheed	Employee	Modify	None
Rasheed	Employee	Grant	None
Bola	Employee	Read	Salary < 50,000
Sola	PurchaseOrder	Insert	Total < 1,000
Sola	PurchaseOrder	Modify	Total < 1,000
PayrollClerks	Employee	Read	None

- What happens as the number of subjects and objects grows?
- Presently, **no commercial DBMS support this level of authorization flexibility.**
- Typically, a DBMS supports some basic authorization models. Application developers must provide more complex constraint enforcement.

i. Subject-Based Security

a. Subjects are individually defined in the DBMS and each object and action is specified.
b. For example, user Salim (a Subject) has the following authorizations:

	Objects			
Actions	**EMPLOYEES**	**ORDERS**	**PRODUCTS**	**...**
Read	Y	Y	Y	...
Insert	N	Y	N	...
Modify	N	Y	Y	...
Delete	N	N	N	...
Grant	N	N	N	...

ii. Object-Based Security

a. Objects are individually defined in the DBMS and each subject and action is specified.
b. For example, the EMPLOYEES table (an Object) has the following authorizations:

	Subjects				
Actions	**SALIM**	**JOHN**	**GAFAR**	**DBA**	**...**
Read	Y	Y	N	Y	...
Insert	N	N	N	Y	...
Modify	N	N	N	Y	...
Delete	N	N	N	Y	...
Grant	N	N	N	Y	...

3.2 The SQL GRANT and REVOKE Statements

SQL provides two main statements for setting up authorization namely: grant and revoke.

3.2.1 SQL Grant Statement

GRANT is used to grant an *action* on an *object* to a *subject*. The SQL syntax is:

```
GRANT action1, action2 ...
ON    object1, object2, ...
TO    subject1, subject2 ...
```

Another option of the GRANT statement is WITH GRANT OPTION. This allows the grantee to propagate the authorization to another subject.

Example:

Let us assume that we have a database with:

Tables: employees, departments, orders, products
Users: salim, john, gafar, dba

```
GRANT INSERT, DELETE, UPDATE, SELECT
ON    employees, departments
TO    salim ;

GRANT INSERT, SELECT
ON orders
TO salim;
```

```
GRANT SELECT
ON products
TO salim;


GRANT SELECT
ON    employees, departments
TO    john;


GRANT INSERT, DELETE, UPDATE, SELECT
ON    employees, departments, orders, products
TO    dba
WITH GRANT OPTION ;
```

Grants can also be done on specific columns in a table:

```
GRANT  SELECT   ON products         TO jones ;

GRANT  UPDATE   ON products (price) TO jones ;
```

If no GRANT statement is issued, it is assumed that no authorization is given (e.g., user GREEN above).

3.2.1 SQL Revoke Statement

REVOKE is used to revoke authorizations from subjects. The SQL syntax is:

```
REVOKE    action1, action2, ...
ON        object1, object2, ...
FROM      subject1, subject2, ...
```

Example:

If Mr. Salim leaves the company, then we should revoke the privileges given to him as follows:

```
REVOKE INSERT, DELETE, UPDATE, SELECT
ON     employees, departments
FROM   salim;

REVOKE INSERT, SELECT   ON orders FROM salim ;

REVOKE SELECT ON products FROM salim;
```

Many RDBMS have an ALL PRIVILEGES option that will revoke all of the privileges on an object from a subject:

```
REVOKE ALL PRIVILEGES
ON     employees, departments, orders, products
FROM   salim ;
```

Activity B

1. Explain SQL commands with examples:
   a. Grant statement
   b. Revoke statement
2. Let us assume that we have a database with the following information:

Tables: employees, departments, orders, products
Users: salim, john, gafar, dba

Write SQL statement that:
   a. Grant read access to salim on employees table
   b. Grant read, insert, update access on all tables to Mr Gafar
   c. Grant modify access on orders table to Mr. John

4.0 Conclusion

Database security is the system, processes, and procedures that protect a database from unintended activity. Unintended activity can be categorized as authenticated misuse, malicious attacks or inadvertent mistakes made by authorized individuals or processes.

Databases provide many layers and types of information security, typically specified in the data dictionary, including: Access control, Auditing, Authentication, Encryption, and Integrity controls

5.0 Summary

In this unit, we have learnt:

lxv. Database security refers to the protection of data against unauthorized access, alteration, or destruction.
lxvi. Access to data should be restricted and should be protected from accidental destruction.
lxvii. In a multi-users environment, database security is needed in order to maintain the consistency of the data.
lxviii. Some of the goals of Database security include confidentiality, integrity, and availability
lxix. Security threat can be classified into accidental and intentional, according to the way they occur.
lxx. Database security can be classified into physical and logical security.
lxxi. The database security system stores authorization rules and enforces them for each database access.
lxxii. Authorization rules take into account a few main ideas such as: Subjects, Objects, Actions, and Constraints.
lxxiii. **GRANT** is used to grant an *action* on an *object* to a *subject*

lxxiv. **REVOKE** is used to revoke authorizations from subjects

6.0 Tutor Marked Assignment

1. Use the table below to answer the following questions:
    a. Write SQL statement that grant authorization to database users shown in the table
    b. Write SQL statement that revoke authorization granted to Mr. Sola

Subject	Object	Action	Constraint
Rasheed	Employee	Read	None
Rasheed	Employee	Insert	None
Rasheed	Employee	Modify	None
Rasheed	Employee	Delete	None
Bola	Employee	Read	Salary < 50,000
Sola	PurchaseOrder	Insert	Total < 1,000
Sola	PurchaseOrder	Modify	Total < 1,000
Chidi	Employee	Read	None

2. How would you ensure database security at Design and Maintenance levels?

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.
w3schools.com (2009). SQL Tutorial. Retrieved April 10th, 2009, from: http://www.w3schools.com/sql/

Module 2: Structured Query Language and Transaction Management

Unit 5: Database Architectures

1.0 Introduction

The database architecture is the set of specifications, rules, and processes that dictate how data is stored in a database and how data is accessed by components of a system. It includes data types, relationships, and naming conventions. The database architecture describes the organization of all database objects and how they work together. It affects integrity, reliability, scalability, and performance. The database architecture involves anything that defines the nature of the data, the structure of the data, or how the data flows.

This unit is intended to be a fairly comprehensive description of database architectures.

2.0 Objectives

What you will learn in this unit is listed below:

a. Database System Architectures
    i. Mainframe Architecture
    ii. Stand-Alone PC Architecture
    iii. File Sharing Architecture
    iv. Classic Client/Server Architecture
    v. Three-Tier Client/Server Architecture
b. Open Database Connectivity
    i. ODBC Implementation
    ii. ODBC Example

3.0 Database System Architectures

There are a number of database system architectures presently in use. One must examine several criteria:

a. Where do the data and DBMS reside?
b. Where are the application program executed (e.g., which CPU)? This may include the user interface.
c. Where are rules of business (applications logic) enforced?

3.1 Traditional Mainframe Architecture

Figure 11.1 is a block diagram of the mainframe architecture. Some of the properties of traditional mainframe database system architecture are:

a. Database (or files) resides on a mainframe computer.
b. Applications are run on the same mainframe computer. e.g., COBOL programs or JCL scripts that access the database.
c. Business rules are enforced in the applications running on the mainframe.

d. Multiple users access the applications through simple terminals (e.g., IBM 3270 terminals or VT220 terminals) that have no processing power of their own. User interface is text-mode screens.
e. Example: DB2 database and COBOL application programs running on an IBM 390.

3.1.1 Advantages of Traditional Mainframe Architecture

a. Excellent security and control over applications
b. High reliability - years of proven MF technology
c. Relatively low incremental cost per user (just add a terminal)

3.1.2 Disadvantages of Traditional Mainframe Architecture

a. Unable to effectively serve advanced user interfaces
b. Users unable to effectively manipulate data outside of standard applications

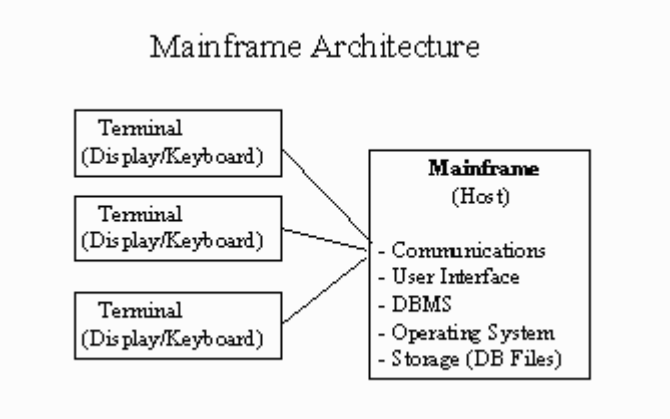


Figure 11.1 Mainframe Architecture
Source: http://cisnet.baruch.cuny.edu/holowczak/

3.2 Personal Computer - Stand-Alone Database

This is referred to as single-user database mode. In the single-user database mode, you can create any number of databases on local or network drives for individual use. This framework is suitable for users who wish to maintain private databases of personal or corporate information, but without losing the ability to easily exchange data with other users' private databases, or with central corporate databases.

Figure 11.2 is a block diagram of the stand-alone architecture. Some of the properties of stand-alone database system architecture are:

a. Database (or files) resides on a PC - on the hard disk.
b. Applications run on the same PC and directly access the database. In such cases, the application *is* the DBMS.
c. Business rules are enforced in the applications running on the PC.

d. A single user accesses the applications.
e. Example: MS Access running on a PC.

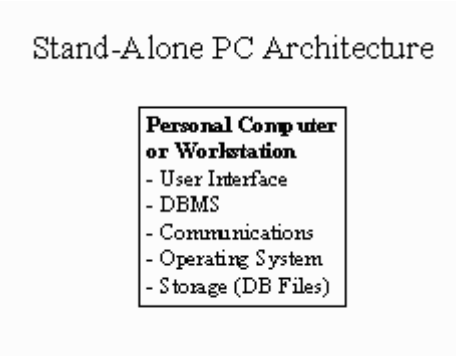


Figure 11.2 Stand-Alone PC Architecture
Source: http://cisnet.baruch.cuny.edu/holowczak/

3.3 File Sharing Architecture

Figure 11.3 is a block diagram of the file sharing architecture. Some of the properties of stand-alone database system architecture are:

a. PCs are connected to a local area network (LAN).
b. A single file server stores a single copy of the database files.
c. PCs on the LAN map a drive letter (or volume name) on the file server.
d. Applications run on each PC on the LAN and access the same set of files on the file server. The application is also the DBMS.
e. Business rules are enforced in the applications - Also, the applications must handle concurrency control. Possibly by file locking.
f. Each user runs a copy of the same application and accesses the same files.
g. Example: Sharing MS Access files on a file server.

3.3.1 Advantages of File Sharing Architecture

a. (limited) Ability to share data among several users
b. Costs of storage spread out among users
c. Most components are now commodity items - prices falling

3.3.2 Disadvantages of File Sharing Architecture

a. Limited data sharing ability - a few users at most

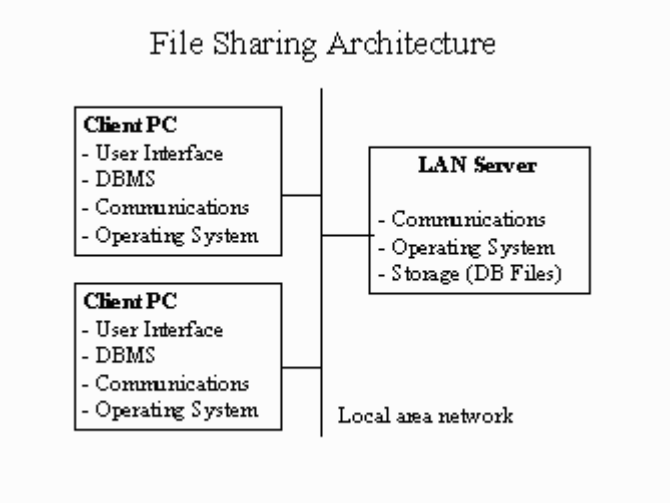


Figure 11.3 File Sharing Architecture
Source: http://cisnet.baruch.cuny.edu/holowczak/

3.4 Two-Tier Client/Server Architecture

A database server is the computer software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server.

Two-tier architecture is one that is familiar to many of today's computer users. A common implementation of this type of system is that of a Microsoft Windows based client program that accesses a server database such as Oracle or SQL Server. Users interact through a GUI (Graphical User Interface) to communicate with the database server across a network via SQL (Structured Query Language).

In two-tier architectures it is important to note that two configurations exist. A thin-client (fat-server) configuration exists when most of the processing occurs on the server tier. Conversely, a fat-client (thin-server) configuration exists when most of the processing occurs on the client machine.

Another example of two-tier architecture can be seen in web-based database applications. In this case, users interact with the database through applications that are hosted on a web-server and displayed through a web-browser such as Internet Explorer. The web server processes the web application, which can be written in a language such as PHP or ASP. The web application connects to a database server to pass along SQL statements which in turn are used to access, view, and modify data. The DB server then passes back the requested data which is then formatted by the web server for the user.

Although this appears to be a three-tier system because of the number of machines required to complete the process, it is not. The web-server does not normally house any of the business rules and therefore should be considered part of the client tier in partnership with the web-browser.

Figure 11.4 is a block diagram of the client-server architecture. Some of the properties of client-server database system architecture are:

a. **Client machines**:

   i. Run own copy of an operating system.
   ii. Run one or more applications using the client machine's CPU, memory.
   iii. Application communicates with DBMS server running on server machine through a *Database Driver*
   iv. Database driver (middleware) makes a connection to the DBMS server over a network.
   v. Examples of clients: PCs with MS Windows operating system. Forms and reports developed in: PowerBuilder, Centura, MS Access, Borland Delphi, Oracle Developer/2000, MS Visual Basic, "C" or "C++", etc.

b. Server Machines:

   i. Run own copy of an operating system.
   ii. Run a Database Management System that manages a database.
   iii. Provides a *Listening* daemon that accepts connections from client machines and submits transactions to DBMS on behalf of the client machines.
   iv. Examples: Sun Sparc server running UNIX operating system. RDBMS such as Oracle Server, Sybase, Informix, DB2, etc.
   PC with Windows NT operating system.

c. **Middleware**:

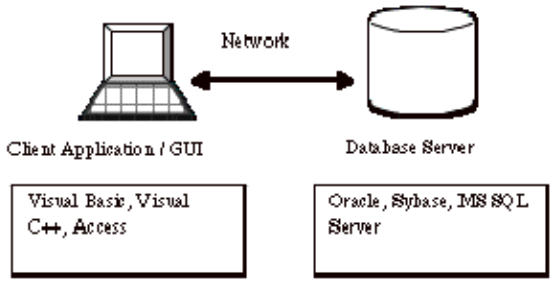
   i. Small portion of software that sits between client and server.
   ii. Establishes a connection from the client to the server and passes commands (e.g., SQL) between them.

3.4.1 Advantages of client/server

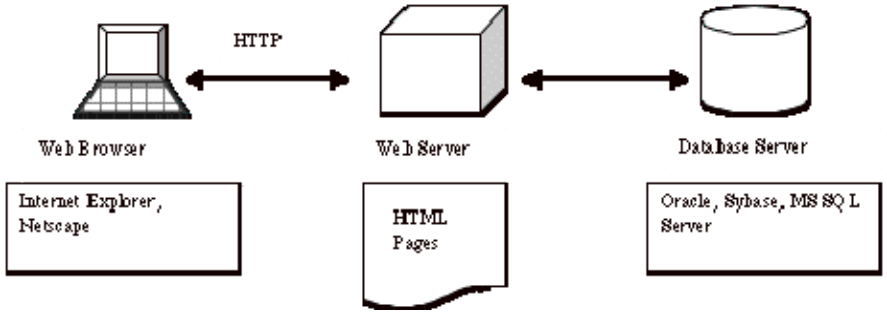
a. Processing of the entire Database System is spread out over clients and server.
b. DBMS can achieve high performance because it is dedicated to processing transactions (not running applications).
c. Client Applications can take full advantage of advanced user interfaces such as Graphical User Interfaces.

3.4.2 Disadvantages of client/server:

a. Implementation is more complex because one needs to deal with middleware and the network.
b. It is possible the network is not well suited for client/server communications and may become saturated.
c. Additional burden on DBMS server to handle concurrency control, etc.



Two-Tier Client-Server Architecture



Web-Based, Two-Tier Client-Server Architecture

Figure 11.4 Two-Tier Client/server Architecture
Source: http://www.windowsecurity.com/

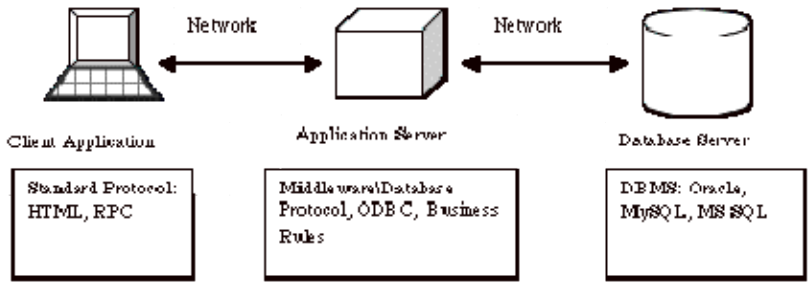
3.5 N-Tier Client/Server Architectures

Most n-tier database architectures exist in a three-tier configuration. In this architecture the client/server model expands to include a middle tier (business tier), which is an application server that houses the business logic. This middle tier relieves the client application(s) and database server of some of their processing duties by translating client calls into database queries and translating data from the database into client data in return. Consequently, the client and server never talk directly to one-another.

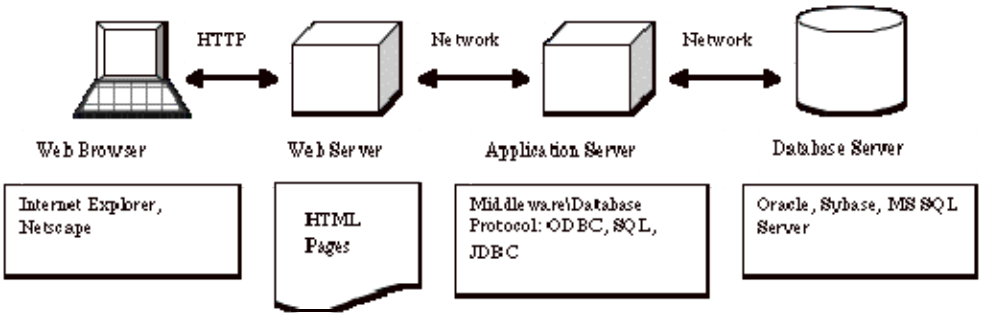
A variation of the n-tier architecture is the web-based n-tier application. These systems combine the scalability benefits of n-tier client/server systems with the rich user interface of web-based systems (see figure 11.5).

Because the middle tier in three-tier architecture contains the business logic, there is

greatly increased scalability and isolation of the business logic, as well as added flexibility in the choice of database vendors.



Three-Tier Client-Server Architecture



Web-Based, Three-Tier Client Server Architecture

Figure 11.5 Three-Tier Client/server Architecture
Source: http://www.windowsecurity.com/

3.5.1 Advantages N-Tier Client/server Architecture

a. Centralize applications logic (one place to make changes)
b. Relieves clients from having to load up on applications logic (the "fat client" approach).
c. Frees up DBMS server to efficiently process transactions

3.5.2 Disadvantages Three-Tier Client/server Architecture

a. System complexity - extremely complex to program and debug
b. Security issues

Activity A

1. What do you understand by the term Database Architecture?
2. Explain the following terms:
   a. Mainframe Database Architecture
   b. Stand-Alone Database Architecture
   c. Two-Tier Database Architecture
   d. Three-Tier Database Architecture

3.6 Open Database Connectivity (ODBC)

Open Database Connectivity (ODBC) is Microsoft's strategic interface for accessing data in a heterogeneous environment of relational and non- relational database management systems. ODBC provides an open, vendor- neutral way of accessing data stored in a variety of proprietary personal computer, minicomputer, and mainframe databases.

ODBC alleviates the need for independent software vendors and corporate developers to learn multiple application programming interfaces. ODBC now provides a universal data access interface. With ODBC, application developers can allow an application to concurrently access, view, and modify data from multiple, diverse databases.

ODBC is a specification to which developers write either:

a. An ODBC-enabled front-end or client desktop application, also known as an ODBC Client.
b. This is the application that the computer-user sees on the computer screen.
c. An ODBC Driver for a "back-end" or "server" DBMS (Database Management System). This is the DBMS application that resides on a computer that is used to store data for access by several users. This application is not what is loaded on the end user's computer. This server application is usually more robust (faster, with centralized security, and backups of data, and so forth) than the client application. The ODBC Driver resides between the ODBC Client and the DBMS; however, it is loaded on the front-end computer.

To use ODBC, client, driver and server components are required

3.6.1 ODBC CLIENT

An ODBC-enabled front-end (also called ODBC client) is one of the required components to use ODBC. Examples of ODBC clients are:

a. Microsoft Access, an application created with Access,
b. An application created with Microsoft Visual Basic
c. An application created with C+Win SDK+ODBC SDK, or ODBC-enabled applications from other vendors (such as Lotus).

3.6.2 ODBC Driver for the ODBC Server

This ODBC driver is software that resides on the front-end. The ODBC Driver Catalog contains an extensive listing of ODBC Drivers. For example, the Microsoft ODBC Driver Pack is a collection of seven ODBC Drivers ready to be used or bundled with ODBC clients. A SQL Server ODBC Driver is included with Access.

3.6.3 DBMS Server

Any ODBC client can access any DBMS for which there is an ODBC Driver. DBMS SERVER is a back-end or server DBMS, for example SQL Server, Oracle, AS/400, Foxpro, Microsoft Access, or any DBMS for which an ODBC driver exists.

3.6.4 How do these three components interact?

The ODBC client uses a language or vocabulary of commands (which is referred to as "ODBC") to request data from, or to send data to, the back- end or server DBMS. However, the DBMS does not understand the ODBC client request until the command passes through the ODBC Driver for that specific DBMS. The ODBC driver translates the command into a format that the ODBC Server can understand. The ODBC Server sends the answer back to the ODBC Driver, which translates the answer into a format that the ODBC Client can understand.

3.6.5 What's so great about ODBC?

- First, application developers do not need to modify their applications to allow them to access data from several back-ends. As long as there is an ODBC Driver for a particular back-end, an ODBC-enabled front-end can access it.
- Second, one ODBC Driver for a particular DBMS allows any ODBC-enabled application to be an ODBC client.

3.6.7 ODBC Implementation

Look in the Control Panel, select administrative tools (shown in figure 11.6, from Windows XP):

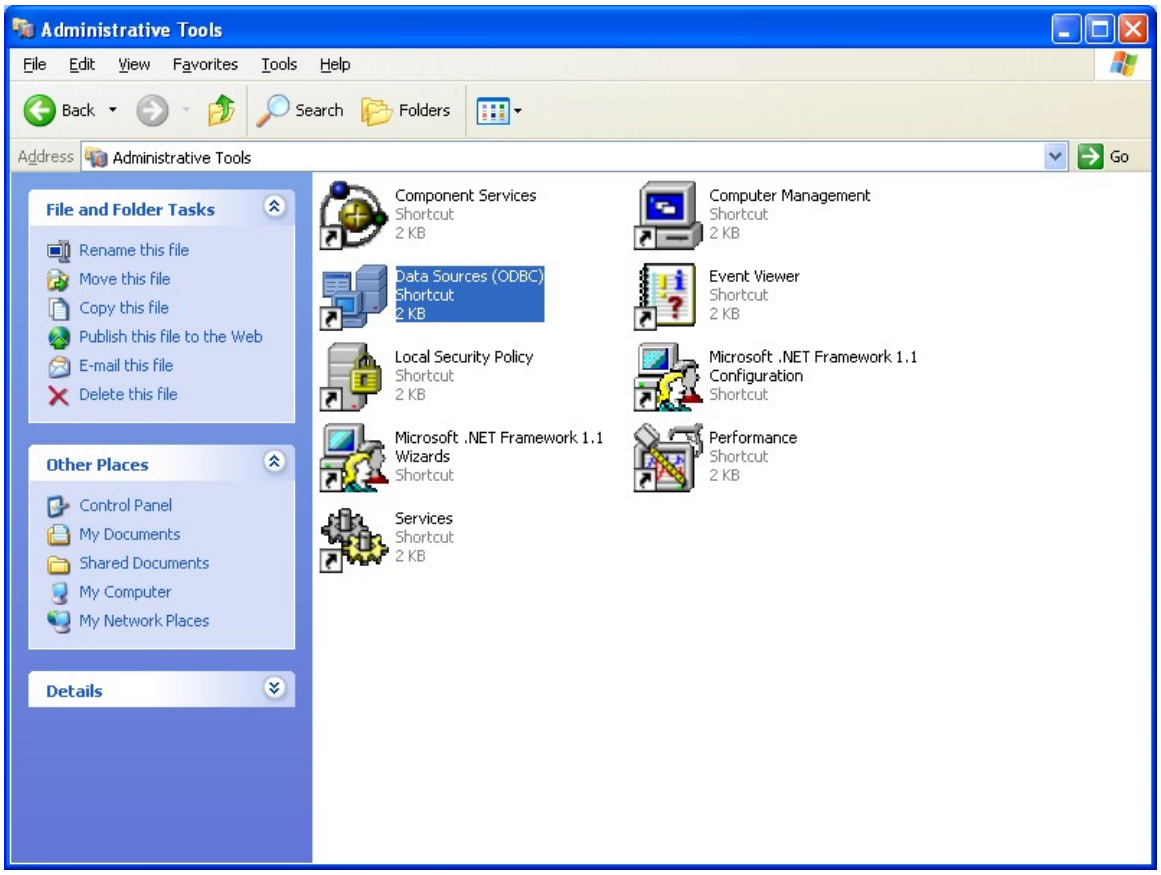


Figure 11.6 Administrative Tools
Source: Microsoft Windows XP Control Panel

Open up the **Data Sources ODBC** icon. This is called the *ODBC Data Source Administrato*.

Click on the **ODBC Drivers** tab to see which drivers are installed:

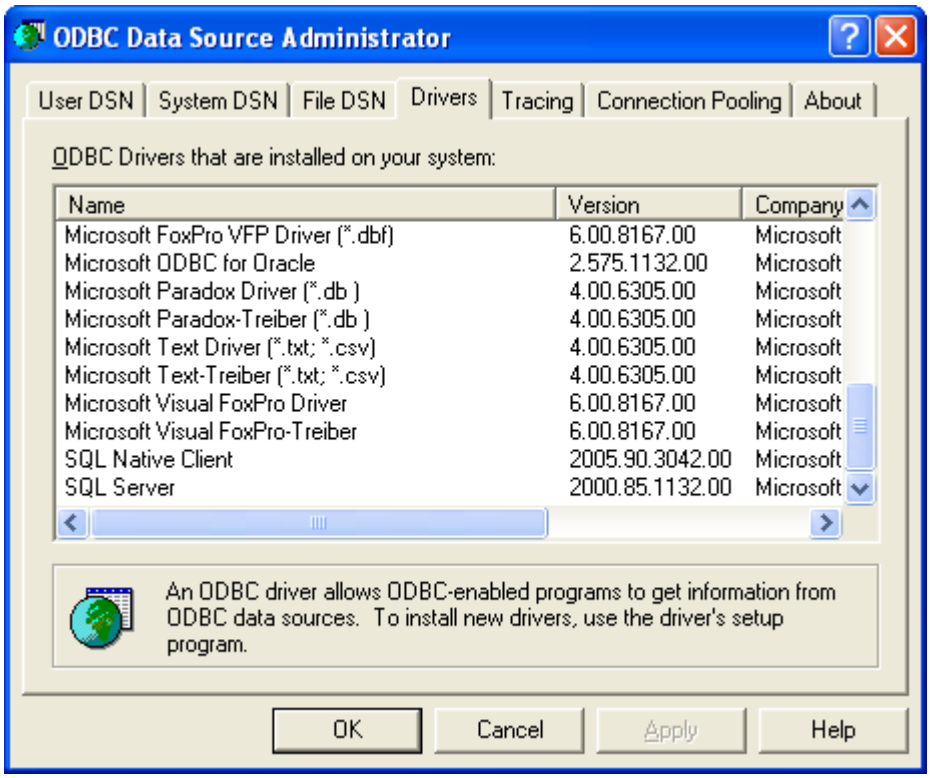


Figure 11.7 ODBC Data Source Administrator
Source: Microsoft Windows XP Control Panel

In the above example (see figure 11.7), we have ODBC drivers for:

Microsoft Access, MS Excel, MS Foxpro, MS Visual FoxPro
Borland dBase and Paradox
Text files
Oracle 8
MS SQL Server

To add more drivers, download or install the ODBC driver from the database manufacturer. The ODBC driver will then appear on this list.

Clicking on the **User DSN** tab shows those data sources that have been defined for a user (see figure 11.8).

A user data source simply gives a name to a configuration that includes:
The specific ODBC driver to be used. In some cases, the specific database table or file that will be used. If necessary, a username and password required to gain access to the database. Finally, any other options the particular ODBC driver requires

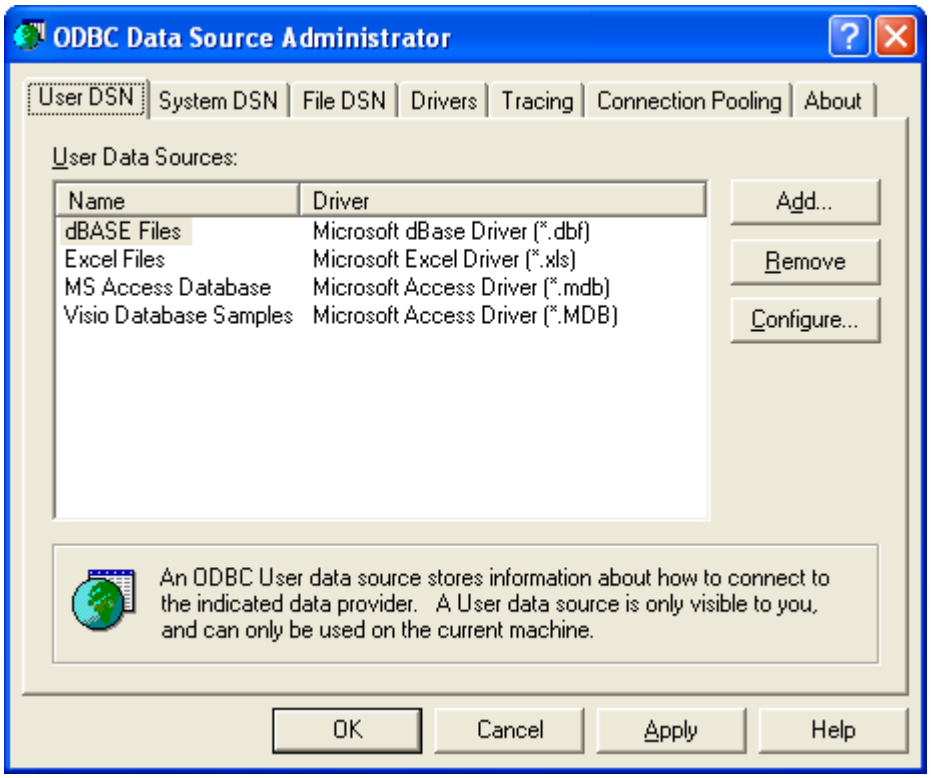


Figure 11.8 User DSN
Source: Microsoft Windows XP Control Panel

User DSNs may only be used by the current user

Systems DSNs may be used by anyone with an account on the computer system

Both User and System DSNs are maintianed in the registry of the local machine.

File DSNs store all of the DSN information in a file that can be shared between users of many machines. e.g., put the File DSN on a file server.

Figure 11.9 shows the setup dialog for a Microsoft Access ODBC driver:

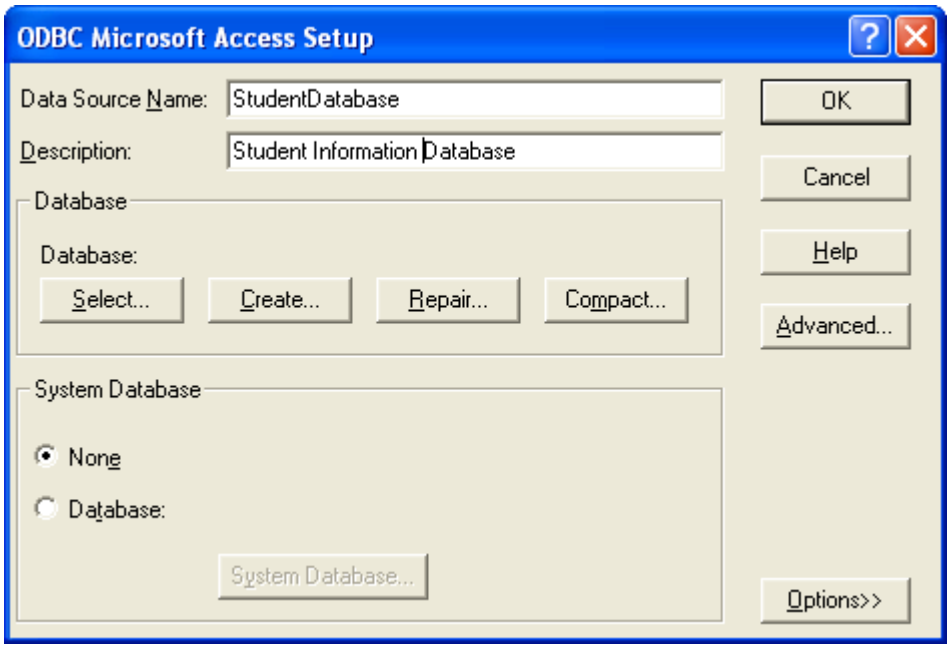


Figure 11.9 Setup dialog for Microsoft Access ODBC Driver
Source: Microsoft Windows XP Control Panel

Activity B

1. Explain the following terms:
   a. ODBC
   b. ODBC Client
   c. ODBC Driver
   d. DBMS Server
2. Explain how to setup an ODBC for SQL Server

4.0 Conclusion

Database architectures can be distinguished by examining the way application logic is distributed throughout the system. Application logic consists of three components: Presentation Logic, Processing Logic, and Storage Logic.

By determining which tier(s) these components are processed on we can get a good idea of what type of architecture and subtype we are dealing with.

5.0 Summary

In this unit, we have learnt:

lxxv. In stand-alone or one-tier database architecture, program (e.g. Microsoft Access) runs on the user's local machine, and references a file that is stored on that machine's hard drive, thus using a single physical resource to access and process information.

lxxvi. Mainframe Architecture is another form of one-tier database architecture. In this system, large machines provide directly connected unintelligent terminals with the means necessary to access, view and manipulate data.

lxxvii. Common implementation two-tier database architecture is that of a Microsoft Windows based client program that accesses a server database such as Oracle or SQL Server.

lxxviii. Two configurations exist in two-tier architectures namely: A thin-client (fat-server) configuration and fat-client (thin-server).

lxxix. In the n-tier configuration, the client/server model expands to include a middle tier (business tier), which is an application server that houses the business logic.

lxxx. A variation of the n-tier architecture is the web-based n-tier application

lxxxi. The middle tier in three-tier architecture contains the business logic.

lxxxii. Open Database Connectivity (ODBC) is Microsoft's strategic interface for accessing data in a heterogeneous environment of relational and non- relational database management systems

lxxxiii. With ODBC, application developers can allow an application to concurrently access, view, and modify data from multiple, diverse databases.

lxxxiv. To use ODBC, client, driver and server components are required

6.0 Tutor Marked Assignment

1a. what do you understand by the term database architectures?

1b. Explain the following database terms

i. Application Logic
ii. One-tier Database Architecture
iii. Two-tier Database Architecture
iv. Three-tier Database Architecture

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall

Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.

Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman

Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Module 3: Design and Development of Database Applications

Unit 1: Introduction to Microsoft Access Tables

1.0 Introduction

This unit introduces fundamental relational database management system (RDBMS) concepts using Microsoft Access, providing a foundation for creating simple tables.

2.0 Objectives

On successful completion you will be able to apply these new skills:

c. Create tables and setting properties and constraints.
d. Create table relationships (One-to-One, One-to-Many, Man-to-Many).

3.0 Database Design Steps

This section serves as a reminder to what we have learnt from all the previous units. The basic steps to design a database are as follows:

i. Collect data about the business model.
ii. Divide the data into main subjects, so called Entities (Entities become tables in most cases).
iii. Assign fields to the subjects, ensuring that the field truly adds information to the subject (attribute of an Entity).
iv. Test the first normal form (1NF)
v. Determine primary keys.
vi. Test the second normal form (2NF) (applies only to composite primary key tables).
vii. Test the third normal form (3NF).
viii. Create relationships:
    a. If a one-to-many relationship is encountered, everything is fine.
    b. If a one-to-one relationship is encountered, question it. Can both tables be combined into one table?
    c. If a many-to-many relationship is encountered, create an intersection table.
ix. Enter some test data and verify that all data is stored accordingly.
x. Fine-tune the database, formats, input masks, validation rules, etc.

3.1 Planning a Microsoft Access Application

To design a **Microsoft Access Database application**, you will first need to define the purpose of the application by determining how it will be used and what the results that it must produce are. You can gather this information by talking to the people who will be using the application. You will want to list the tasks that the users must perform with the database and gather together examples of the current paper forms and reports that they use and produce.

After analyzing the database users need and workflow, you can then decide how users would be able to navigate through your application and complete their tasks. Some of the

customized navigation tools that you can incorporate into your application design include command buttons, custom menu commands and custom toolbar buttons. You can also design an interface that controls how the database applications will start-up and what parts of the database application are available to individuals or groups of the database users. Other elements of the graphical user interface that you should consider are the layout that you should use, how you will group particular objects and the logic that you will apply to allow the user to move from one object to another.

As you will be designing database queries, forms, reports and other objects based upon your database table design, it is extremely important that you take time up front to plan a sound database structure and its relationships. As you analyze the data that the users will be working with, separate it into different subjects, each of which will become an entity. You can eliminate data redundancy and inconsistent data dependency by normalizing your data to ensure that all tables are in at least third normal form.

You will also need to plan the database security. Planning security means that you can control what individuals or groups of users can do with the database tables, queries, forms, reports, macros and modules. You will need to determine, again by interviewing the application's users, who should have access to an application's objects and data and who should be able to change an object's design.

3.2 Creating a Microsoft Access Application

Once you have worked through the stages of Planning a Microsoft Access Application, you will then move onto creating the application in Microsoft Access. The following checklist details the application needs and data sources:

a. Investigation Phase

i. Talk to the users who will be working with the database application to find out their data input needs, reporting needs, querying and other data needs and application security needs.
ii. Create a rough prototype by using the Database Wizard and other wizards and templates.

b. Planning the Database Tables

i. Account for all of the data.
ii. Normalise the data tables.
iii. Set up the database relationships and referential integrity.
iv. Create tables, write in field descriptions in design view and add sample data to test the design.

c. Planning the Queries

i. Create queries for forms and reports.

ii. Create queries for selecting and modifying data.

d. Create the required Forms

i. Start with the forms needed for data entry.
ii. Test the forms.
iii. Test the working prototype by using macros to automate tasks.

e. Creating the necessary Reports

i. Start with the reports needed to display entered information.
ii. Test the reports.

f. Create other Forms or Reports

i. Create forms or reports for other user designated uses.
ii. Test the forms and reports.

g. Connecting and automating the Tables, Forms and Reports

i. Create buttons, menus and toolbars needed for navigation purposes.
ii. Test the buttons, menus and toolbars

h. Verifying Application Design with the client

i. Verify that forms, reports and queries perform as desired.
ii. Verify that all data is accounted for.
iii. Discuss any additional features that may be required.

i. Adding the final Application Design touches

i. Create a splash screen for the Microsoft Access application.
ii. Create a start-up form for the application.
iii. Add captions.
iv. Refine the design of the forms.
v. Convert macros to Access VBA (Visual Basic for Applications)
vi. Set startup properties

j. Implementing Security

i. Create user and group security dependant upon who can access what in the database application.
ii. Assign permissions to the security groups.
iii. Make Backups of the application.

k. Testing, Rollout and Training Users of the Microsoft Access Application

i. Test the application to verify that everything works as required.
ii. Roll out the application to the end users.
iii. Plan a training program to train the users of the application. Consider the depth of documentation appropriate to distribute to the trainers and the users.

3.3 Introduction to Microsoft Access

Microsoft Access is a powerful program to create and manage your databases. It has many built in features to assist you in constructing and viewing your information.

First of all you need to understand how Microsoft Access breaks down a database. Some keywords involved in this process are: Database File, Table, Record, Field, Data-type.

Database File: This is your main file that encompasses the entire database and that is saved to your hard-drive or floppy disk. Example is Bank.mdb

Table: A table is a collection of data about a specific topic. There can be multiple tables in a database. Examples: Customers and Account tables

Field: Fields are the different categories within a Table. Tables usually contain multiple fields. Example: Customers LastName, Customers FirstName etc

Data types: Data types are the properties of each field. A field only has one data type. Example: the Lastname field in student table could be of type Text

3.3.1 Starting Microsoft Access

There are two ways to this:

i. Double click on the Microsoft Access icon on the desktop (see figure 12.1).

Figure 12.1: Microsoft Access icon
Source: Microsoft Corporation

ii. Click on Start --> Programs --> Microsoft Access (see figure 12.2)

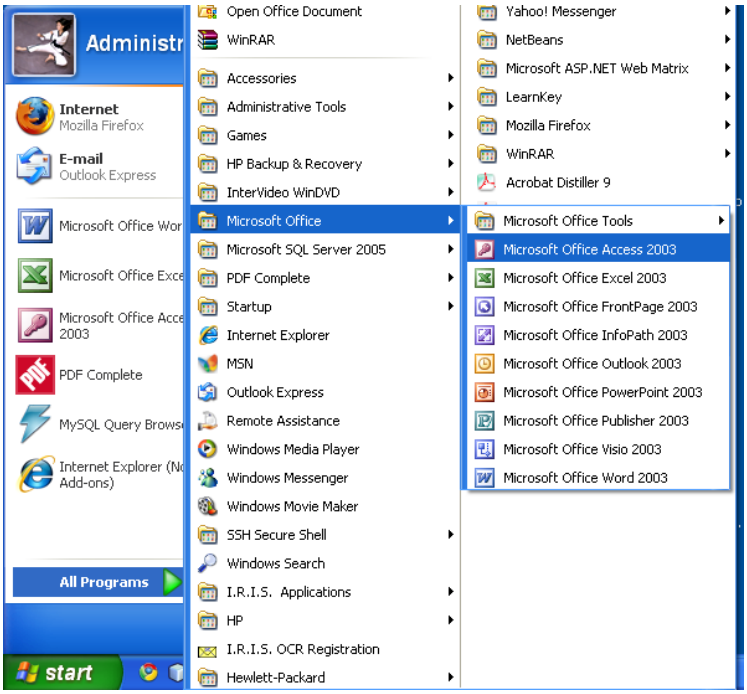


Figure 12.2: Start Microsoft Access window
Source: Microsoft Corporation

3.3.2 Create a database using the Database Wizard

The steps involved are:

a. When Microsoft Access first starts up, a dialog box is automatically displayed with options to create a new database or open an existing one. If this dialog box is displayed, click **Access Database Wizards, pages, and projects** and then click **OK.**

   If you have already opened a database or closed the dialog box that displays when Microsoft Access starts up, click **New Database** on the toolbar.

b. On the **Databases** tab, double-click the icon for the kind of database you want to create.

c. Specify a name and location for the database.

Click **Create** to start defining your new database

3.3.3 Create a database without using the Database Wizard

The steps involved are:

a. When Microsoft Access first starts up, a dialog box is automatically displayed with options to create a new database or open an existing one. If this dialog box is displayed, click **Blank Access Database**, and then click **OK**.

If you have already opened a database or closed the dialog box that displays when Microsoft Access starts up, click **New Database** on the toolbar, and then double-click the **Blank Database** icon on the **General** tab.

b. Specify a name and location for the database and click **Create**. (Figure 12.3is the screen that shows up following this step)

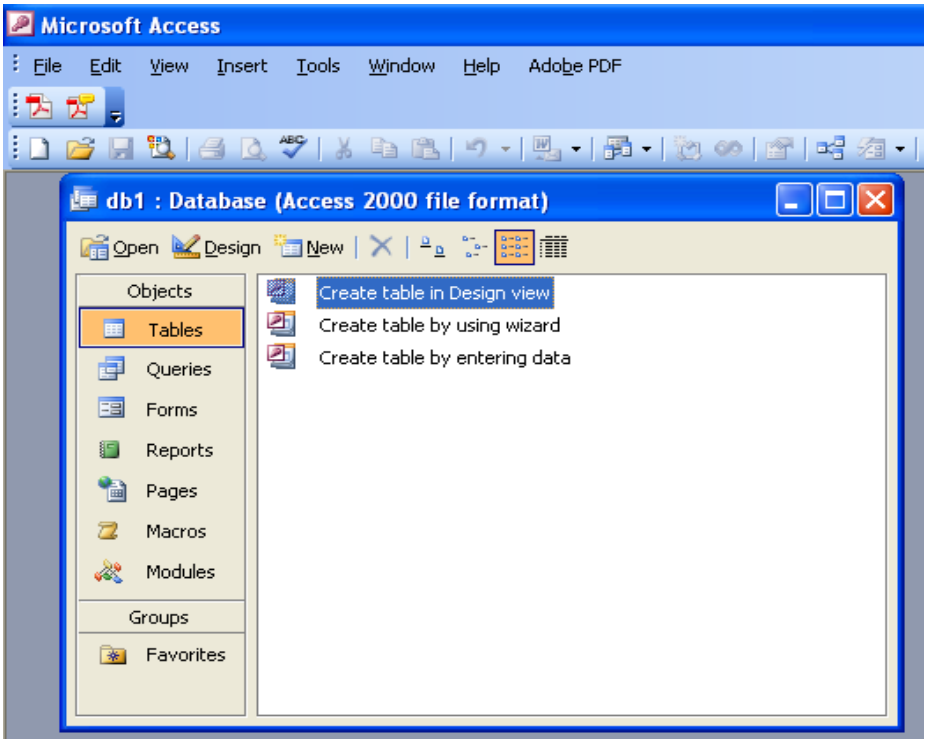


Figure 12.3: Microsoft Access Database Screen
Source: Microsoft Corporation

The two main features of this main screen are the menu bar that runs along the top of the window and the series of *tabs* in the main window. The menu bar is similar to other Microsoft Office products such as Excel. The menus include:

i. File - Menu items to Open, Close, Create new, Save and Print databases and their contents. This menu also has the Exit item to exit Access.
ii. Edit - Cut, Copy, Paste, Delete
iii. View - View different database objects (tables, queries, forms, reports)
iv. Insert - Insert a new Table, Query, Form, Report, etc.
v. Tools - A variety of tools to check spelling, create relationships between tables, perform analysis and reports on the contents of the database.
vi. Window - Switch between different open databases.
vii. Help - Get help on Access.

The tabs in the main window for the database include:

i. Tables - Displays any tables in the database.
ii. Queries - Displays any queries saved in the database.
iii. Forms - Displays any forms saved in the database.
iv. Reports - Displays any reports saved in the database.
v. Macros - Displays any macros (short programs) stored in the database.
vi. Modules - Displays any modules (Visual Basic for Applications procedures) stored in the database.

3.4 Creating Tables in Microsoft Access

Tables are the main units of data storage in Access. There are a number of ways to create a table in Access. Access provides *wizards* that guide the user through creating a table by suggesting names for tables and columns. The other main way to create a table is by using the *Design View* to manually define the columns (fields) and their data types.

3.4.1 Creating a Table Using the Design View

To create a table in Access using the Design View, perform the following steps:

i. Click on the Tables tab in the left hand pane of the database dialog box
ii. Double click on the "Create Table in Design View" item in the right hand pane.

The Table Design View will appear. Fill in the **Field Name**, **Data Type** and **Description** for each column/field in the table. Refer to table 7.2, Module 2, Unit 1 for available data types in Microsoft Access.

3.4.2 Primary Key

One or more fields (columns) whose value or values uniquely identify each record in a table. A primary key does not allow Null values and must always have a unique value. A primary key is used to relate a table to foreign keys in other tables.

NOTE: You do not have to define a primary key, but it is usually a good idea. If you do not define a primary key, Microsoft Access asks you if you would like to create one when you save the table.

To define primary key, simply select the field or fields to be used and select the primary key button (see figure 12.4)

Figure 12.4: Primary Key button
Source: Microsoft Corporation

3.4.3 Switching Views

To switch views form the datasheet (spreadsheet view) and the design view, simply click the button in the top-left hand corner of the Access program.

Datasheet view button allows you to enter raw data into your database table (see figure 12.5)

Figure 12.5: Datasheet button
Source: Microsoft Corporation

Design view button allows you to enter fields, data-types, and descriptions into your database table (see figure 12.6)

Figure 12.6: Datasheet button
Source: Microsoft Corporation

3.4.4 Entering Data

Click on the Datasheet View and simply start entering the data into each field (see figure 12.7). **NOTE:** Before starting a new record, the primary key field must have something in it.

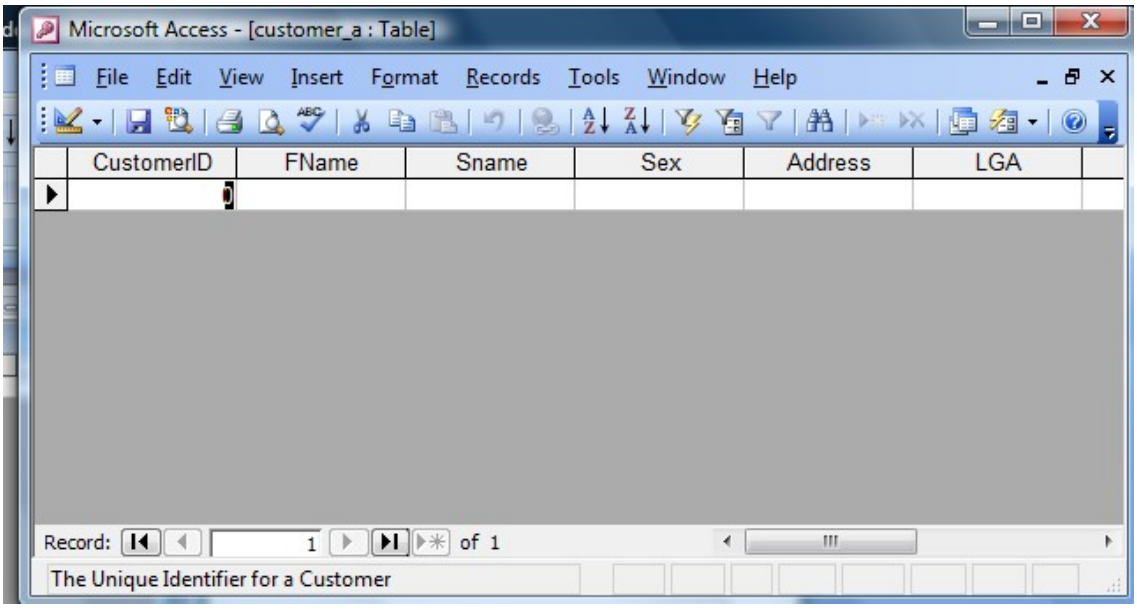


Figure 12.7: Data entry screen
Source: Microsoft Corporation

3.4.5 Manipulating Data

To add a new row, simply drop down to a new line and enter the information

To update a record, simply select the record and field you want to update, and change its data with what you want

To delete a record, select the entire row and hit the Delete Key on the keyboard

Activity A

1. Create a database application for bank customer information.
   a. Start Microsoft Access
   b. Create a blank database and save it as bank.mdb
   c. Create Customer table with the following fields
      Fill in the information for the fields as follows:

Field Name	Data Type	Description
CustomerID	Number	The Unique Identifier for a Customer
FName	Text	The First Name of the Customer
Sname	Text	The Surname of the Customer
Sex	Text	Sex of Customer
Address	Text	Customer ‘s Address
LGA	Text	Customer’s Local Government of Residence
State	Text	The State of residence of the Customer

   d. Choose CustomerID field as the Primary key
   e. Create account table with the following fields

Field Name	Data Type	Description
CustomerID	Number	The Unique Identifier for a Customer
AccountNumber	Number	The Unique Identifier for a Bank Account
AccountType	Text	The type of account (Checking, savings, etc.)
DateOpened	Date	The date the account was opened
Balance	Number	The current balance (money) in this account (in $US)

   f. Choose AccountNumber as the Primary key
   g. Open the customer table enter the following data
   h. Open the account table and enter the following data

CustomerID	FName	Sname	Sex	Address	LGA	State
1001	Odusanya	Adebimpe	Male	23A Adetokunbo Str.	Ibadan North	Oyo
1002	Nkechi	Francis	Female	12 Mavis Ave.	Nnewi East	Enugu
1003	Bala	Usman	Male	44 Kano Way	Mantu	Kaduna
1004	Christopher	Bello	Male	661 Parker Rd.	Etiosa	Lagos
1005	Patricia	Popoola	Female	23 Queens Road	Ikoyi	Lagos

CustomerID	AccountNumber	AccountType	DateOpened	Balance
1001	9987	Current	10/12/1989	4000.00
1001	9980	Savings	10/12/1989	2000.00
1002	8811	Savings	01/05/1992	1000.00
1003	4422	Current	12/01/1994	6000.00
1003	4433	Savings	12/01/1994	9000.00
1004	3322	Savings	08/22/1994	500.00
1004	1122	Current	11/13/1988	800.00

3.5 Creating Relationships between Tables

After you have set up multiple tables in your Microsoft Access database, you need a way of telling Access how to bring that information back together again. The first step in this process is to define relationships between your tables.

A relationship works by matching data in key fields - usually a field with the same name in both tables. In most cases, these matching fields are the primary key from one table, which provides a unique identifier for each record, and a foreign key in the other table.

In the Bank database we have created, the Customers table is related to the Accounts table by virtue of the CustomerID field appearing in both tables.

Steps involved in creating relationship:

i. On the menu bar click on Tools --> Relationships. The Show Table dialog box will appear as shown in figure 12.8

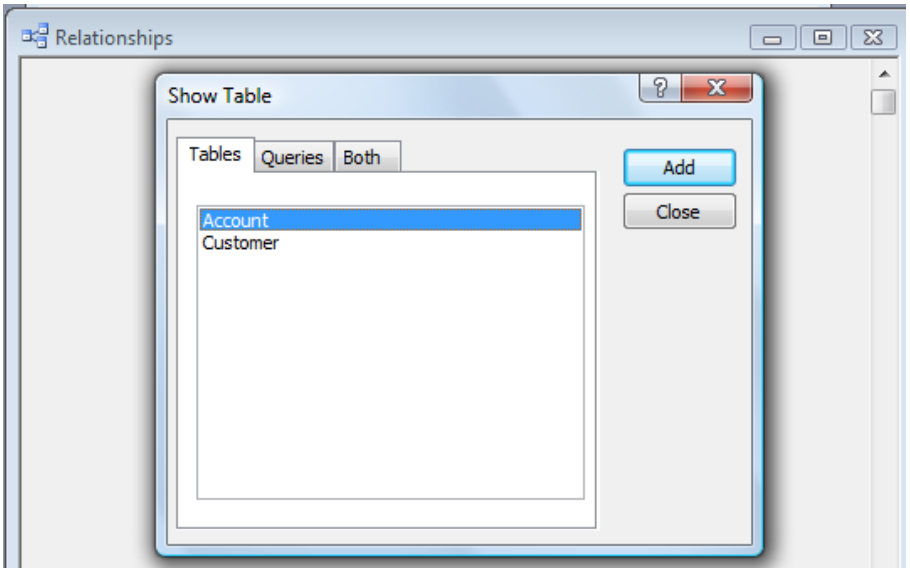


Figure 12.8: Relationship Screen
Source: Microsoft Corporation

ii. Highlight both the Customers table and the Accounts table as shown below and then click on the `Add` button.

iii. Click on the `Close` button to close this dialog box. The Relationships screen will now reappear with the two tables displayed in figure 12.9

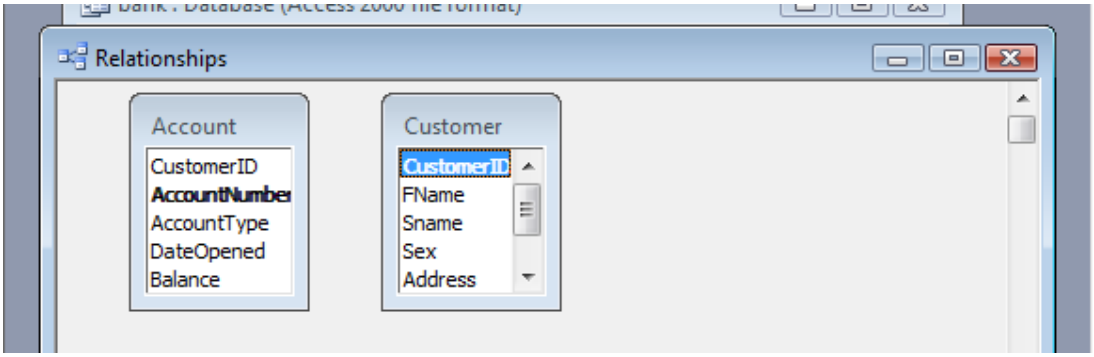


Figure 12.9: Relationship Screen with tables
Source: Microsoft Corporation

iv. To connect the Customers table with the Accounts table to form a relationship, click on the CustomerID field in the Customers table and drag it over on top of the CustomerID field on the Accounts table. Upon releasing the mouse button, the Edit Relationships dialog box will appear as in figure 12.10

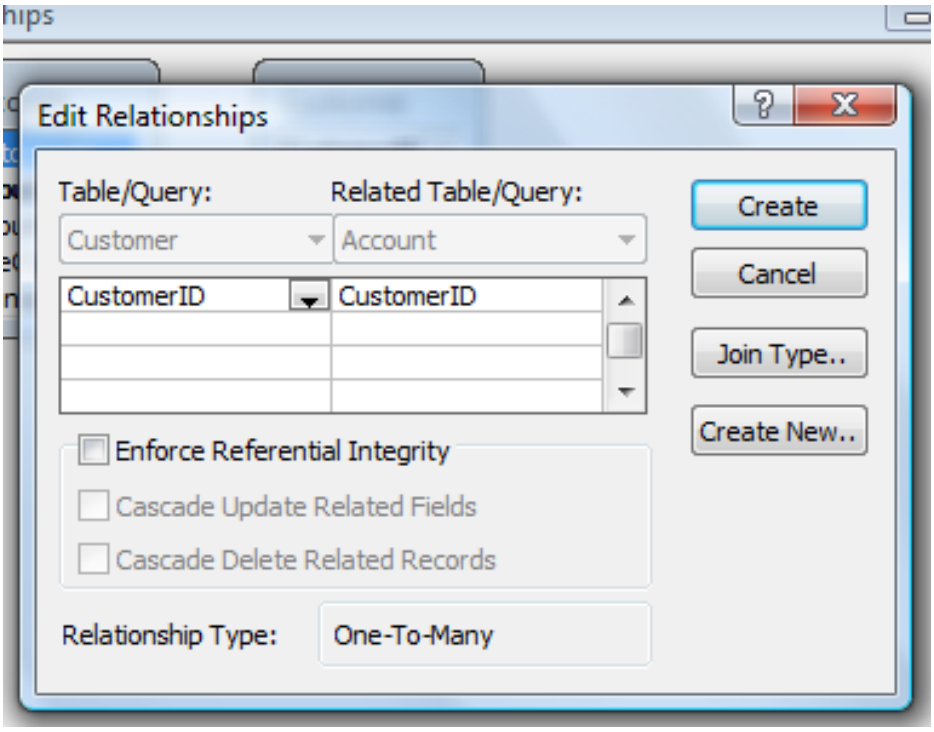


Figure 12.10: Edit Relationship Screen
Source: Microsoft Corporation

Access tries to determine the Relationship Type between the two tables. Most times two tables have a *One-to-Many* relationship and this is usually the default chosen by Access. For this example, Access knows that CustomerID is a key of the Customer table so it chooses this field as the "One" side. This makes the Accounts table the "Many" side as *One* customer may have *Many* accounts.

v. One additional step to be taken is the check off the box labeled "Enforce Referential Integrity". This option puts constraints into effect such that an Accounts record can not be created without a valid Customer and Access will also prevent a user from deleting a Customer record if a related Accounts record exists. At this point, click on the `Create` button to create the relationship. The Relationships screen should reappear with the new relationship in place as shown if figure 12.11
    a. When the Cascade Update Related Fields check box is set, changing a primary key value in the primary table automatically updates the matching value in all related records.
    b. When the Cascade Delete Related Records check box is set, deleting a record in the primary table deletes any related records in the related table

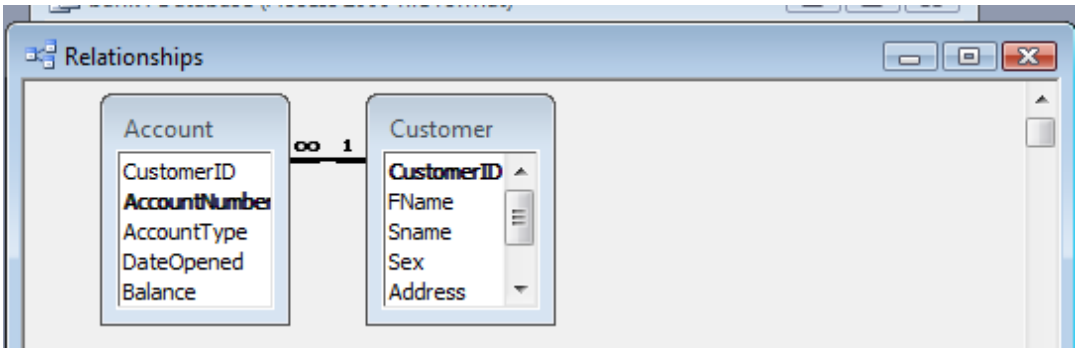


Figure 12.11: New Relationship Screen
Source: Microsoft Corporation

Note the symbols "1" (indicating the "One" side) and the infinity symbol (indicating the "Many" side) on the relationship. Close the relationships screen and select `Yes` to save the changes to the Relationships layout.

Activity B

1. Explain how to create relationship between tables in a database
2. What do you understand by the term Referential Integrity?
3. How would you enforce referential integrity in a relationship?

4.0 Conclusion

Microsoft Access is a very powerful relational database management tool for creating database applications. It has many built in features to assist you in constructing and viewing your information.

5.0 Summary

In this unit, we have learnt:

lxxxv. Microsoft Access is a powerful program to create and manage your databases.
lxxxvi. Database file is the main file that encompasses the entire database.
lxxxvii. Tables are the main units of data storage in Microsoft Access.
lxxxviii. Tables can be created by using design view or by using wizards
lxxxix. Fields are the different attributes of a Table.
xc. Data types are the properties of each field and a field only has one data type.
xci. Primary key is one or more fields (columns) whose value or values uniquely identify each record in a table.
xcii. Before starting a new record, the primary key field must have something in it.
xciii. It necessary to define relationships between tables in a database
xciv. A relationship works by matching data in key fields in both tables.
xcv. Referential integrity option in relationship puts constraints into effect.

xcvi. When the Cascade Update Related Fields check box is set, changing a primary key value in the primary table automatically updates the matching value in all related records.

xcvii. When the Cascade Delete Related Records check box is set, deleting a record in the primary table deletes any related records in the related table.

6.0 Tutor Marked Assignment

1. What are the basic steps in designing a database application?

2. Examine the following flat file and design the relational model for this kind of a data:

ISBN	Title	AuthorID	Author Name	AuthorPhone	PubID	Publisher Name	Publisher Phone	Price
1	Iliad	3	Iyabo	01-234-7777	1	Mint Press	020-234-9870	N500
2	Freedom	8	Olu	02-345-9988	1	Mint Press	020-324-6789	N500
3	Love	7	Ayo	09-589-0081	1	Mint Press	020-324-6790	N1000
4	Indy	1	Ade	04-589-0082	1	Mint Press	020-456-7890	N600
5	C++	4	Ken	06-589-0083	1	Mint Press	020-456-7890	N1500
6	Visual J++	4	Ken	03-589-0084	1	Mint Press	020-456-7890	N750
7	A+	6	Sulia	08-589-0085	2	Ajayi Press	234-999-9999	N1200
8	Bello	5	Chidi	09-589-0086	2	Ajayi Press	234-999-9999	N300
9	Bus	5	Chidi	02-589-0087	2	Ajayi Press	234-999-9999	N250
10	Mill	5	Chidi	01-589-0088	2	Ajayi Press	235-999-9999	N1800
11	Booking	2	Ray	01-589-0089	3	Ann Press	210-000-0000	N1800
12	Alade	1	Ade	01-589-0090	3	Ann Press	210-000-0000	N1800
13	Bukky	11	Kate	02-589-0091	3	Ann Press	210-000-0000	N1500
14	Bukky	12	Salim	05-589-0092	3	Ann Press	210-000-0000	N1500
15	Bukky	13	Slim	06-589-0093	3	Ann Press	210-000-0000	N1500
16	Broad	9	John	07-589-0094	3	Ann Press	210-000-0000	N750
17	Broad	10	Mba	09-589-0095	3	Ann Press	210-000-0000	N750

7.0 Further Reading and other Resources

cisnet.baruch.cuny.edu (2008). Microsoft Access Tutorial. Retrieved January 15th, 2008, from http://cisnet.baruch.cuny.edu/holowczak/classes/2200/access/accessall.html
databasedev.co.uk (2003-2006). Create Database Applications using Microsoft Access, Retrieved October 10th, 2008, from: http://www.databasedev.co.uk/plan-an-access-application.html
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Microsoft.com (2009). Microsoft Access help file. Retrieved March 15th, 2009 from http://microsoft.com/office/access/default.htm.
Microsoft.com (2009). Microsoft Access Tutorial: Retrieved March 15th, 2009 from http://www.bcshool.net/staff/accesshelp.htm

Module 3: Design and Development of Database Applications

Unit 2: Introduction to Microsoft Access Queries

1.0 Introduction

Queries are very useful tools when it comes to databases and they are often called by the user through a form. They can be used to search for and grab data from one or more of your tables, perform certain actions on the database and even carryout a variety of calculations depending on your needs.

In this unit, we will use Access to create a variety of queries that analyze and manipulate database information.

2.0 Objectives

On successful completion you will be able to:

e. Create a variety of queries that analyze and manipulate database information.

3.1 Types of Microsoft Access Queries

Microsoft Access allows for many types of queries, some of the main ones being select, action, parameter and aggregate queries. Table 13.1 shows different types of queries in Microsoft Access

Table 13.1: Types of Queries
Source: **http://www.brainbell.com/tutorials/ms-office/Access_2003/**

Query Type	Description
	The most basic and common type of query, select queries find and display the data you want from one or more tables or queries.
	Prompts the user for specific information every time the query is run.
	Summarizes data in a table format that makes it easy to read and compare information.

While select queries display information that matches your criteria, the following action queries do something to the data that matches your criteriasuch as change or delete it.

	Creates a new table from all or part of the data in one or more tables. Useful for backing up and exporting information.
 Append Query	Appends or adds selected records from one table to another table. Useful for importing information into a table.
 Delete Query	Deletes selected records from one or more tables.
 Update	Updates selected information in a table. For example, you could raise the

Query Type	Description
Query	prices on all trips to Europe by 15 percent.
Union Query	Combines fields from two or more tables or queries into one field and is written directly in SQL.

3.1.1 Select Query

The select query is the simplest type of query and because of that, it is also the most commonly used one in Microsoft Access databases. It can be used to select and display data from either one table or a series of them depending on what is needed.

In the end, it is the user-determined criteria that tell the database what the selection is to be based on. After the select query is called, it creates a "virtual" table where the data can be changed, but at no more than one record at a time.

3.1.2 Action Query

When the action query is called, the database undergoes a specific action depending on what was specified in the query itself. This can include such things as creating new tables, deleting rows from existing ones and updating records or creating entirely new ones.

Action queries are very popular in data management because they allow for many records to be changed at one time instead of only single records like in a select query.

There are four kinds of action queries:

a. Append Query – takes the set results of a query and append (or add) them to an existing table.
b. Delete Query – deletes all records in an underlying table from the set results of a query.
c. Make Table Query – as the name suggests, it creates a table based on the set results of a query.
d. Update Query – allows for one or more field in your table to be updated.

a. Creating Append Query

To Create an Append Query:

i. Create a new query, select Design view, and click OK.
ii. Click the tables and/or queries you want to use in the append query, click Add, and then click Close when you are through.
iii. Click the Query Type button list arrow on the toolbar and select Append Query or select Query ⟶ Append Query from the menu.
iv. Select the table to which you want to add the results of the query.

v. If you select an existing table, click one of the following options: Current Database (if the table is in the currently open database) or Another Database (and type the name of the other database, including the path, if necessary). Click OK, and then add the fields you want to append and identify a matching field if Access does not supply one.
vi. Click OK and click the View button on the toolbar to view the results of the query or the Run button on the toolbar to append the records.

b. Creating Delete Query

To Create a Delete Query:

i. In the Database window, click the Queries tab in the Objects bar and click the New button.
ii. Select Design view and click OK.
iii. Add the appropriate tables and/or queries and click close, and then connect any unrelated tables.
iv. Click the Query Type button list arrow on the toolbar and select Delete Query.
v. Click the View button to view the results of the delete query.
vi. If you are satisfied that the appropriate records will be deleted, click the Run button on the toolbar and click yes to confirm the deletion.

c. Creating a Make-Table Query

To Create a Make-Table Query:

i. In Design view, create a select query; including tables and fields.
ii. Click the Query Type button list arrow on the toolbar and select Make-Table Query Type the name of the table you want to create. If you select an existing table, click one of the following options: Current Database (if the table is in the currently open database) or Another Database (and type the name of the other database, including the path, if necessary).
iii. Click OK.
iv. Click the View button on the toolbar to view the results of the query or the Run button on the toolbar to create the new table.

d. Creating an Update Query

To Create an Update Query:

i. Create a new query in Design view, and then select the tables and/or queries you want to use in the update query.
ii. Click the Query Type button list arrow on the toolbar and select Update Query or select Query Update Query from the menu.
iii. Double-click the fields that you want to appear in the query or click and drag the fields onto the design grid.

iv. Enter an expression to update the selected field and enter any criteria, if needed, to select which records should be updated.
v. Click the View button to view the results of the update query. If you're satisfied that the appropriate records will be updated, click the Run button on the toolbar to update the records.

3.1.3 Parameter Query

In Microsoft Access, a parameter query works with other types of queries to get whatever results you are after. This is because, when using this type of query, you are able to pass a parameter to a different query, such as an action or a select query. It can either be a value or a condition and will essentially tell the other query specifically what you want it to do.

It is often chosen because it allows for a dialog box where the end user can enter whatever parameter value they wish each time the query is run. The parameter query is just a modified select query.

3.1.4 Aggregate Query

This is a special type of query. It works on other queries (such as selection, action or parameter) just like the parameter query does, but instead of passing a parameter to another query it totals up the items by selected groups. It essentially creates a summation of any selected attribute in the table.

The SQL aggregate functions available to Microsoft Access are:

a. Sum
b. Avg
c. Min
d. Max
e. First
f. Last
g. Group By
h. Count
i. StDev
j. Var
k. Expression
l. Where

3.2 Creating Select Queries in MS Access

Creating a query can be accomplished by using either the query design view or the Query wizard. In this section, we will use design view.

Queries are accessed by clicking on the **Queries** tab in the Access main screen as shown in figure 13.1:

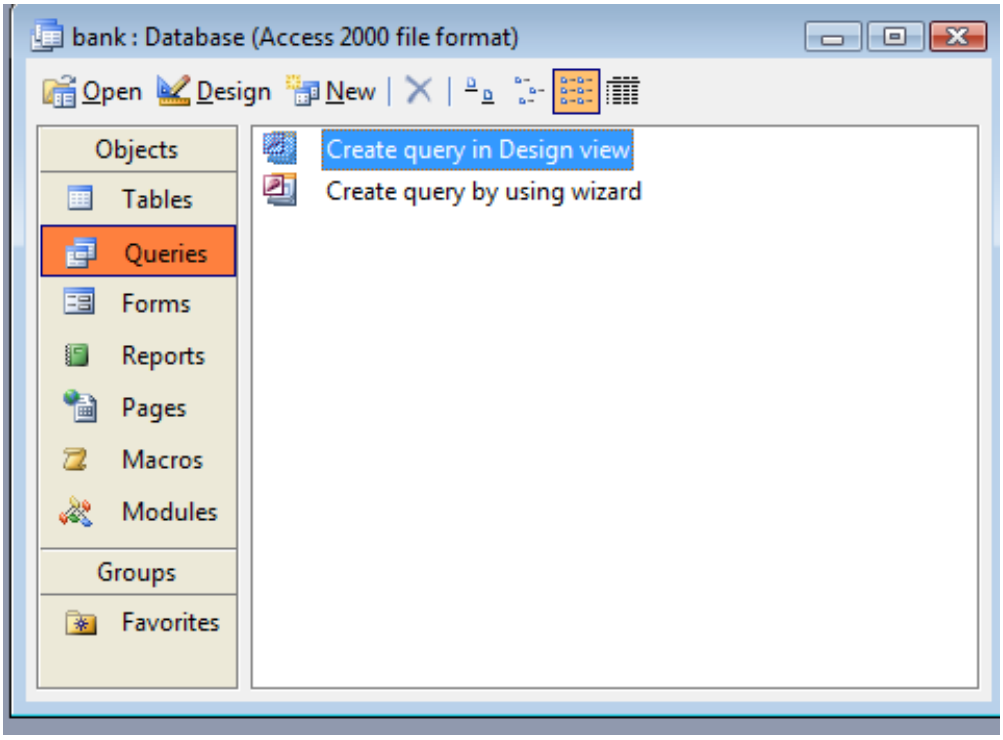


Figure 13.1: Microsoft Access Queries Screen
Source: Microsoft Office

To create a new query in design view:

i. Click the queries icon in the objects bar, then double-click create query in design view.
ii. Select the table or query you want to use and click.
iii. Repeat step 2 as necessary for additional tables or queries. Click close when you are through.
iv. Double-click each field you want to include from the field list or drag the field from the field list onto the design grid add (see figure 13.2).

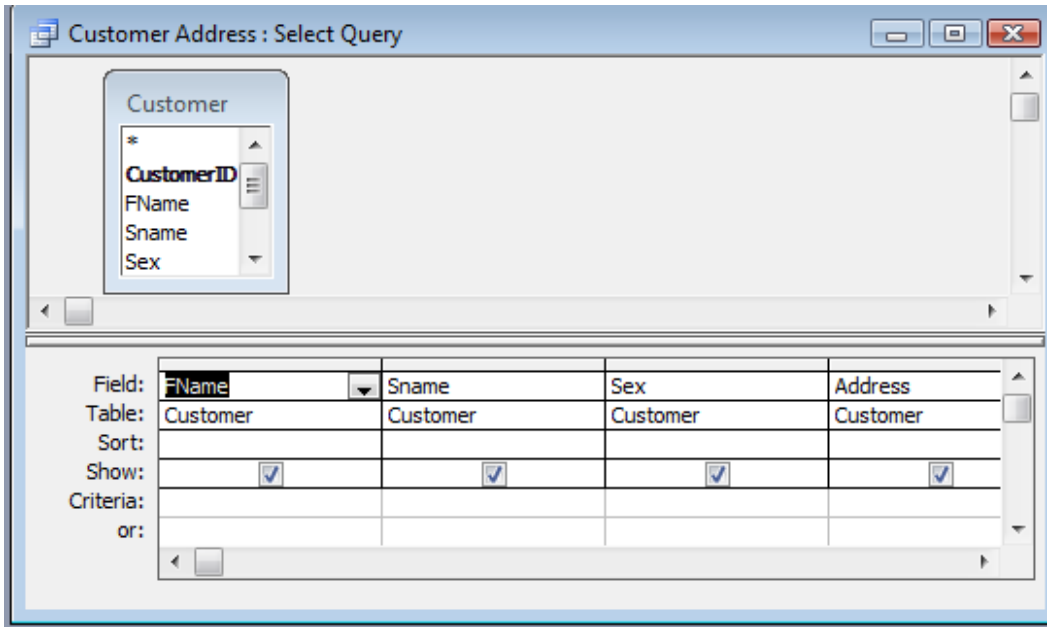


Figure 13.2: Table and field selection
Source: Microsoft Office

v. In the design grid enter any desired criteria for the field in the criteria row (see figure 13.3).

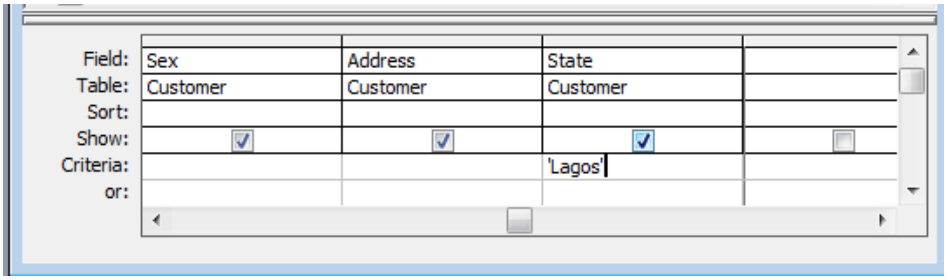


Figure 13.2: Microsoft Access Queries Criteria
Source: Microsoft Office

vi. Click the sort box list arrow for the field and select a sort order (see figure 13.4).

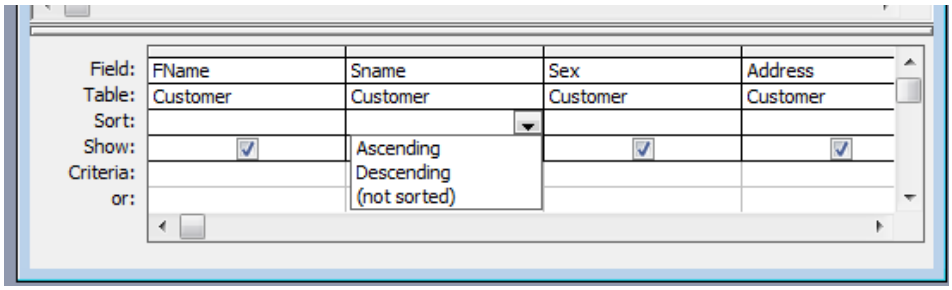


Figure 13.4: Microsoft Access Queries Sort Order
Source: Microsoft Office

vii. Close the query window.
viii. Click yes to save the query, enter a query name, and click OK.

Activity A

1. Display the list of customers that reside in Lagos sorted by their surname.

a. Open the Bank database created in unit 1
b. Click the Queries icon in the Objects bar and then double-click Create query in Design view.
c. Click the Customer table and click Add.
d. Click Close.
e. Double Click each of the fields in the field list.
f. Click the State column's Criteria row and type Lagos.
g. Click the Sname column's Sort box list arrow and select Ascending.
h. Save the query as `qryCustomer` and click OK.
i. Run the query by pulling down the `Query` menu and choosing the `Run` menu item.

What is the output?

3.3 Creating a Calculated Field

A calculated field performs some type of arithmetic on one or more fields in a database to come up with a completely new field.

You must create an expression (or formula) to perform a calculation. To enter fields in an expression, type the field name in brackets ([FName]). If a field name exists in more than one table, you will need to enter the name of the table that contains the field in brackets ([Customer]) followed by an exclamation mark (!). Then type the field name in brackets, such as [FName].For example, customer surname in a customers table could be represented as [Customers]![SName]

To create an expression or calculated in a query:

a. Display the query in design view.
b. Click the field row of a blank column in the design grid.
c. Enter the field name for the calculated field followed by a colon (:).
d. Enter the expression you want Microsoft access to calculate, using the proper syntax
e. Click the view button or run button on the toolbar to see the results.

3.4 Working with IIf Function

Functions are used to create more complicated calculations or expressions than operators can.

There are several hundred functions in Microsoft Access, but all of them are used in a similar way:

The name of the function, followed by the arguments in parenthesis.

An argument in Microsoft Access is the value a function uses to perform its calculation.

This section introduces a very useful database function: the IIf function. The IIf function evaluates a condition and returns one value if the condition is true and another value if the condition is false. The syntax is:

IIf(*expr, truepart, falsepart*)

The **IIf** function syntax has these named arguments:

Part	**Description**
expr	Required. Expression you want to evaluate.
truepart	Required. Value or expression returned if ***expr*** is **True.**
***falsepa*rt**	Required. Value or expression returned if ***expr*** is **False**

To create an IIf (IF...THEN) Function:

a. Display the query in design view.
b. Click the field row of a blank column in the design grid.
c. Enter the field name for field name followed by a colon (:).
d. Enter the expression using the syntax IIf(expr, truepart, falsepart)

3.5 Summarising Group of Records

When you work with queries, you will often be less interested in the individual records and more interested in summarized information about groups of records. A query can calculate information about a group of records in one or more tables. For example, you could create a query that finds the total tuition fee paid by student in all the departments a particular academic. The Total row lets you group and summarize information in a query. The Total row normally is tucked away from view in the query design window you can make the Total appear by selecting View followed by Totals from the menu. Once the Total row is displayed, you can tell Microsoft Access how you want to summarize the fields. Table 13.2 is a summary of the available Total options in Microsoft Access.

Table 13.2: Total Options
Source: **http://www.brainbell.com/tutorials/ms-office/Access_2003/**

Option	Description
Group By	Groups the values in the field so that you can perform calculations on the groups.
Sum	Calculates the total (sum) of values in a field.
Avg	Calculates the average of values in a field.
Min	Finds the lowest value in a field.
Max	Finds the highest value in a field.
Count	Counts the number of entries in a field, not including blank (Null) records.
StDev	Calculates the standard deviation of values in a field.
Var	Calculates the variance of values in a field.
First	Finds the values from the first record in a field.
Last	Finds the values from the last record in a field.
Expression	Tells Access that you want to create your own expression to calculate a field.
Where	Specifies criteria for a field to limit the records included in a calculation.

To summarize a group of records:

a. Display the query in design view.
b. If necessary, click the totals button on the toolbar.
c. Move the field that you want to group data by onto the design grid and make sure group by appear in that field total row.
d. Move the field that you want to perform calculation onto the design grid. Choose the type of calculation that you want to perform.

Activity B

1. Create query using Wizards and design views.

a. Open the Bank database created in unit 1
b. Click the Queries icon in the Objects bar and then double-click Create query using wizard.
c. Select Accounts table from Table/Queries dropdown list.
d. Move the AccountNumber, AccountType and Balance fields over to the selected fields' area. Then click the `Next` button.

e. In the next panel, you will be asked to choose between a detail or summary query. Choose detailed query and click on the `Next` button
f. Save the query with qryAccounts and click on the `Finish` button..

2. Modify the query to sort the ouput on the account number and only display the saving account.

a. From the Queries tab on the Access main screen, highlight the qryAccounts and click on the `Design` button.
b. Change the `Sort` order for the **AccountNumber** field to Ascending.
Add the following statement to the `Criteria under the` **AccountType** field:
`= 'Savings'`
c. Run the query by pulling down the `Query` menu and choosing the `Run` menu item. The output is shown below:
d. Save and close the query to return to the Access main screen.

3. Create multiple tables query

a. Create a new query called "Accounts Summary Query" that joins the Cusomers table (include the CustomerID and Name fields) with the Accounts table (include the Balance field only).
b. In the second step of the wizard, click on the `Summary` choice (instead of Details) and then click on the `Summary Options...` button.
c. Check all of the Summary option boxes such as **Sum**, **AVG**, **Min** and **Max**.

4.0 Conclusion

Queries are a fundamental means of accessing and displaying data from tables. Queries can access a single table or multiple tables.

5.0 Summary

In this unit, we have learnt:

xcviii. Queries can be used to search for and grab data from one or more tables.
xcix. some of the main queries types in Microsoft Access are: select, action, parameter and aggregate queries
c. Select query can be used to select and display data from either one table or a series of them depending on what is needed.
ci. There are four kinds of action queries: Append, Delete, Make-Table, and Update
cii. Parameter query can be used to pass parameter to select and action queries
ciii. Aggregate query essentially creates a summation of any selected attribute in the table.

6.0 Tutor Marked Assignment

1. Which of the following criterion is NOT written using the proper syntax?
    A. "Harris"
    B. Between 1/1/2000 and 12/31/2000
    C. NO VALUE
    D. 500
2. Which of the following types of queries are action queries? (Select all that apply.)
    A. Parameter queries.
    B. Append queries.
    C. Update queries.
    D. Crosstab queries.
3. Which of the following expressions is NOT written in the correct syntax?
    A. [Order Total]*[Tax Rate]
    B. "Order Total"*0.1
    C. [tblCustomerTours]![Cost]*[tblEmployees]![Commission]
    D. 100+10
4. If you are having trouble remembering how to write expressions using the correct syntax, you can use the Expression Builder to help you create the expression. (True or False?)
5. Rebate: IIF([Age]65,"Senior","Adult") This expression is an example of:
    A. Something I learned back in high school algebra and thought I would never see again.
    B. A financial expression.
    C. Something that belongs in a Microsoft Excel book.
    D. A conditional expression.
6. A query prompts a user for a date and then displays only records that contain the specified date. Which type of query is this?
    A. A parameter query.
    B. A crosstab query.
    C. An action query.
    D. An update query.
7. You must create a report if you want to calculate totals for a group of records, as queries can't perform this task. (True or False?)
8. A query summarizes information in a grid, organized by regions and months. Which type of query is this?
    A. A parameter query.
    B. A crosstab query.
    C. An action query.
    D. An update query.
9. Your company finally agreed to buy you a nifty 3COM Palm palmtop. Now you want to extract your clients from the company's database and put them into a separate table that you can export to your Palm. Which type of query could help you accomplish this task?
    A. A parameter query.
    B. A crosstab query.

C. An update query.
D. A make-table query.

10. If you are creating a crosstab query, what must the table you are querying contain?
    A. At least one text field.
    B. At least one number field.
    C. More than 100 records.
    D. Lots of confusing information.
11. How can you add a table to the query design window?
    A. Select Edit ⟶Add Table from the menu.
    B. Click the Show Table button on the toolbar.
    C. Select the table from the Table list on the toolbar.
    D. Select Tools ⟶Add Table from the menu.
12. You want a query to calculate the total sales for your employees. How can you do this from the query design window?
    A. Click the Totals button on the toolbar. In the Total row select "Group By" under the Employee field and "Sum" under the Sales field.
    B. Click in the Sales field and click the AutoSum button on the toolbar.
    C. You need to export this information to Microsoft Excel and calculate it there.

7.0 Further Reading and other Resources

Brainbell.com (2008). Microsoft Access Tutorial. Retrieved June 20th, 2008, from http://www.brainbell.com/tutorials/ms-office/Access_2003/
Bcschool.net (2003-2006). Create Database Applications using Microsoft Access, Retrieved June 20th, 2008, from http://www.bcshool.net/staff/accesshelp.htm
Cisnet.baruch.cuny.edu (2009). Microsoft Access Tutorial. Retrieved March 15th, 2009 from http://cisnet.baruch.cuny.edu/holowczak/classes/2200/access/accessall.html.
Databasedev.co.uk (2009). Microsoft Access Tutorial: Retrieved March 15th, 2009 from http://www.databasedev.co.uk/plan-an-access-application.html

Module 3: Design and Development of Database Applications

Unit 3: Introduction to Microsoft Access Forms

1.0 Introduction

Data entry forms are the primary means of entering data into tables in the database. In the previous units, we described how to add data to a table using a spreadsheet-like view of the data. Data entry forms offer a more user-friendly interface by adding labels for each field and other helpful information.

Microsoft Access provides several different ways of creating data entry forms. These include creating the forms by hand using a Design View as well as a number of wizards that walk the user through the forms creation process.

This unit explains everything you have ever wanted to know about forms.

2.0 Objectives

On successful completion you will be able to:

f. View and use Microsoft Access forms
g. Design forms to present information in any way you like.
h. Combine data from several related tables or queries.

3.0 Introduction to Forms

Microsoft Access provides the tools for developing graphical user interfaces (GUI) that facilitate the use of database applications.

An Access GUI consists of a set of Forms. Forms are front ends for accessing the data that is stored in database tables or that is generated by queries. Some of the available controls are: Text labels, Text boxes, List boxes, Combo boxes, Option groups, Buttons, Objects created by other applications, Decorative lines and boxes.

3.1 Controls

Forms are made up of controls. Individual control is typically “bound” to a particular field of the table or query that is associated with the form. Therefore, a “screen” of a form displays the contents of a record of the associated table/query whereas individual “bound” controls display the values of individual fields within that record.

Every control has a set of properties. Properties determine where a form/control gets its data from, whether the form/control can be used for editing data or for displaying data only as well as several details which determine how the form/control is displayed. Form/control properties are automatically set by “Wizard” programs provided by Access in order to facilitate the creation of forms. Users only need to edit them occasionally, in

order to fine-tune the appearance and behavior of the forms they create. Some of the available properties in Microsoft Access are shown in Table 14.1a and b

Table 14.1a: Common Forms/Report Properties
Source: http://www.brainbell.com/tutorials/ms-office/

Property	Tab	Description
Caption *	Format	Displays a descriptive caption for a form or text label.
Format *	Format	Customizes the way numbers, dates, times, and text are displayed and printed.
Decimal Places *	Format	Determines the number of decimal places displayed.
Visible *	Format	Shows or hides a control. Useful if you want to use information on the form without it being visible. For example, you could use the value in a hidden control as the criteria for a query.
Display When	Format	Determines whether a section or control always appears or only appears when it is displayed on screen or printed.
Scroll Bars	Format	Determines whether scroll bars appear in the control.
Left *	Format	Determines the horizontal position of the control.
Top *	Format	Determines the vertical position of the control.
Width *	Format	Determines the width of a control.
Height *	Format	Determines the height of a control.
Back Style	Format	Determines whether a control is transparent or not.
Back Color	Format	Determines the color of a control. Click the ... button to select a color from a palette.
Special Effect	Format	Applies a 3-D effect to a control.
Border Style	Format	Determines the line style of a control's borderselect from transparent lines, solid lines, dashed lines, etc.
Border Color	Format	Determines the color of a control's border. Click the ... button to select a color from a palette.
Border Width	Format	Determines the width of a control's border (in points).
Fore Color	Format	Determines the color of text in a control or the fill color of an object. Click the ... button to select a color from a palette.
Font Name	Format	Determines the font used in a control (such as Arial or Times New Roman).
Font Weight	Format	Determines the thickness (boldface) of text in a control.
Font Italic	Format	Determines whether the text in a control appears in italics.

Property	Tab	Description
Font Underline	Format	Determines whether the text in a control is underlined.
Text Align	Format	Determines how text should be aligned in a control.
Control Source *	Data	Determines the data that appears in the control.
Input Mask *	Data	Limits the amount and type of information that can be entered in a field, such as (___) ___ _-_____ _ for a phone number. Click the ... button to create an input mask using the Input Mask Wizard.
Default Value *	Data	Specifies a value that is automatically entered in this field for new records.
Validation Rule *	Data	Allows you to enter an expression that is evaluated when data in the field is added or changed.
Validation Text *	Data	Allows you to enter a message that is displayed when data doesn't meet the Validation Rule property.
Locked *	Data	Determines whether changes can be made to a field's data.
Event Tab	Event	Allows you to assign a macro or Visual Basic procedure to a specific event, such as when you click or update a control.
Name *	Other	Specifies the name of the control that identifies it in expressions, macros, and Visual Basic procedures.
Status Bar Text	Other	Specifies a message to display in the Status bar when the control is selected.
Enter Key Behavior	Other	Determines if pressing the Enter key adds a new line of text in a control or if it moves to the next field.
Allow AutoCorrect	Other	Determines if AutoCorrect (i.e., "teh" → "the") is used in a control.
AutoTab	Other	Used with the Input Mask property. Determines whether an automatic tab to the next field occurs when the last character permitted by a text box control's input mask is entered.
Tab Stop	Other	Determines whether users are able to tab to the control.
Tab Index	Other	Determines the tab order.
Shortcut Menu Bar	Other	Specifies a user-created shortcut menu that appears when the control is right-clicked.
ControlTip Message	Other	Specifies a brief message that appears when a user points at the control for a couple of seconds.
Help Context Id	Other	Specifies an identifier number for a user-created Help file that appears when the user selects the control and presses F1.

Property	Tab	Description
Tag	Other	Specifies extra, user-defined information that is stored in the object.

Table 14.1b: Important Forms Properties
Source: http://www.brainbell.com/tutorials/ms-office/

Property	Tab	Description
Caption *	Format	Displays a descriptive caption in the form's title bar.
Default View *	Format	Determines the view the form is in when opened. Single Form: Displays one record at a time. Continuous Forms: Displays multiple records in a form. Datasheet: Displays multiple records in a Datasheet. PivotTable: Dynamically analyzes data, summarizes into a table. PivotChart: Dynamically analyzes data, summarizes into a chart.
Allow Form View	Format	Determines if users can switch to this view.
Allow Datasheet View		
Allow PivotTable View		
Allow PivotChart View		
Scroll Bars *	Format	Determines whether scroll bars appear on the form.
Record Selectors *	Format	Determines whether a form contains a record selector.
Navigation Buttons *	Format	Determines whether a form has navigation buttons.
Dividing Lines	Format	Determines if lines appear between records in continuous forms.
Auto Resize	Format	Resizes the form automatically to display a complete record.

Property	Tab	Description
Border Style *	Format	Determines the type of window the form appears in: None, Thin, Sizable, or Dialog.
Control Box	Format	Determines if a control menu appears in the form.
Min Max Buttons	Format	Determines if minimize and/or maximize buttons appear in the form.
Close Button	Format	Determines if a close button appears on the form.
Width *	Format	Determines the width of the form.
Height *	Format	Determines the height of the form.
Picture	Format	Adds a graphic or picture for the form or report background. Click the Build button to browse for the folder and file.
Picture Type	Format	Determines if the picture is embedded or linked.
Picture Size Mode	Format	Determines how the contents of a picture frame are displayed: Clip, Stretch, or Zoom.
Picture Alignment	Format	Determines the alignment of a picture within a frame.
Picture Tiling	Format	Determines whether a picture is tiled within a frame.
Grid X	Format	Determines the number of subdivisions (horizontal) in a grid.
Grid Y	Format	Determines the number of subdivisions (vertical) in a grid.
Layout for Print	Format	Determines whether the form uses printer fonts.
Palette Source	Format	Specifies the path and file name for the graphic file used as a palette.
Record Source *	Data	Specifies the table or query whose data will be used in the form.
Filter	Data	Specifies a filter that is loaded automatically with the Form/Report.
Order By	Data	Specifies a sort order that is loaded automatically with the Form/Report.
Allow Filters	Data	Determines whether filters may be applied to the form.
Allow Edits *	Data	Determines whether records can be modified in the form.
Allow Deletions *	Data	Determines whether records can be deleted in the form.
Allow Additions *	Data	Determines whether records can be added in the form.
Data Entry *	Data	Allows you to select "Yes" if you only want to use the form to add new records.

Property	Tab	Description
Event Tab	Event	Allows you to assign a macro or Visual Basic procedure to a specific event, such as when you click or update a control.
Pop Up	Other	Determines whether the form appears in a pop-up window that remains on top of all other windows.
Modal	Other	Determines whether the form keeps the focus (you can't switch to any other windows or forms) until it is closed.
Cycle	Other	Determines how the tab key should cycle.
Menu Bar	Other	Allows you to select a custom menu bar that you created that should appear when the form is active.
Toolbar	Other	Allows you to select a custom toolbar that you created that should appear when the form is active.
Shortcut Menu	Other	Determines if right mouse button shortcut menus are permitted in the form.
Shortcut Menu Bar	Other	Specifies a user-created shortcut menu that appears when a user clicks the right-mouse button.
Fast Laser Printing	Other	Print the form using optimized laser-printer formatting.
Help File	Other	Specifies the name of the custom Help file for the form.
Help Context Id	Other	Specifies an identifier number for a user-created Help file that appears when the user selects the control and presses F1.
Tag	Other	Specifies extra user-defined information that is stored in the form.
Has Module	Other	Specifies if the form has Visual Basic code behind it.

3.2 Creating Forms using Forms Wizard

Microsoft Access provides a set of Wizards that facilitate the creation of new forms.

To create a simple form that displays records from a table:

a. Click the Forms tab of the main database window.
b. Choose the **Create form using Wizard**
c. Specify the table/query with which the new form will be associated (for example: Customers). See figure 14.1

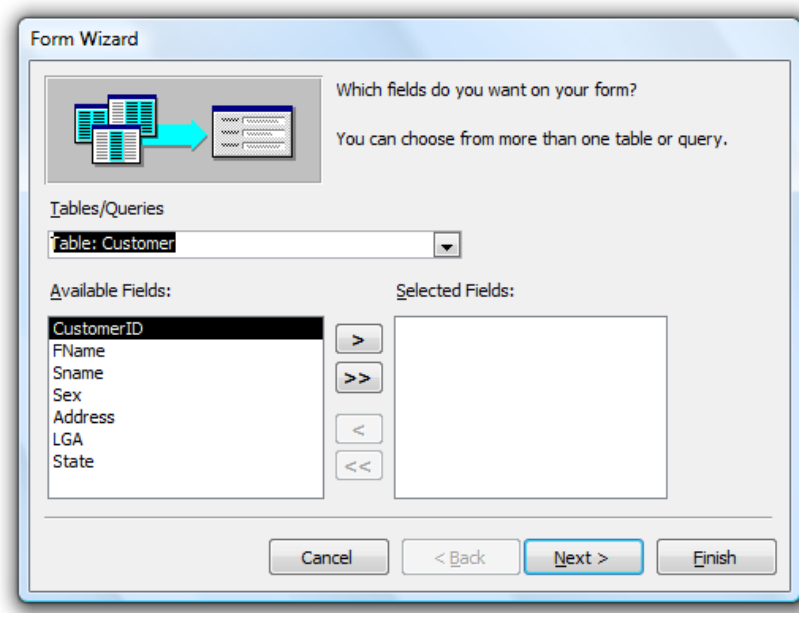


Figure 14.1: Form Wizard Dialog box 1
Source: Microsoft Corporation

d. Select which fields of the selected table/query will actually appear on the form. (See figure 14.2)
e. Click on the Next button
f. Select the general layout of the form (See figure 14.3). The available options are:
    i. Columnar - Places the labels to the left of each field. This is similar to a paper form. This layout is suitable for viewing data one record at a time.
    ii. Tabular - Places the field labels at the top of the screen and the records are displayed below. This is similar to how a spreadsheet would display the data and is suitable for displaying multiple records of data at a time.
    iii. Datasheet - The data appears in the same fashion as when viewing or adding data to a table.
    iv. Justified - Places the labels above each field with the fields spread out on the form. This is suitable for viewing a single record at a time as with the columnar layout.
g. Click on the Next button
h. Choose the desired form background pattern (see figure 14.4).
i. Give name under which the new form will be stored (see figure 14.5).
j. Click Finish Button.
k. Finally, the form will be created and open for data display/editing (see figure 14.6).

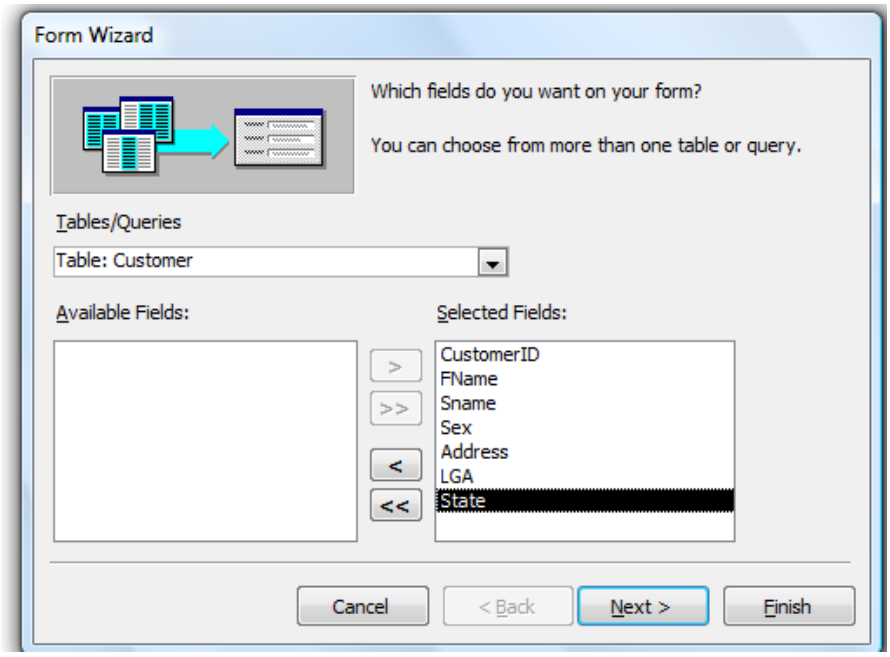


Figure 14.2: Form Wizard Dialog box 2
Source: Microsoft Corporation

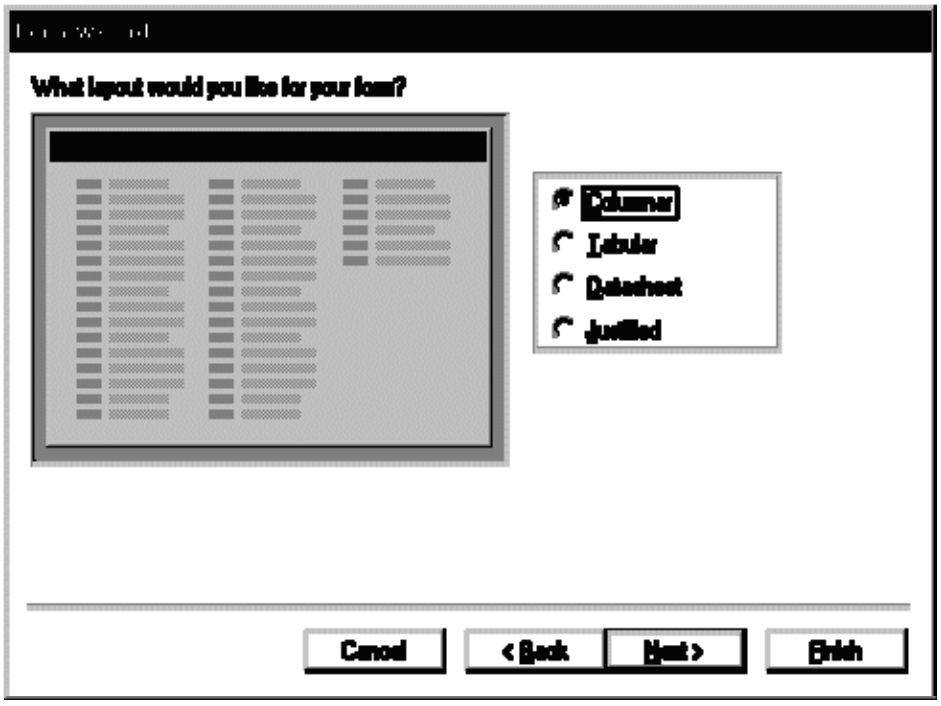


Figure 14.3: Form Wizard Layout Dialog box
Source: Microsoft Corporation

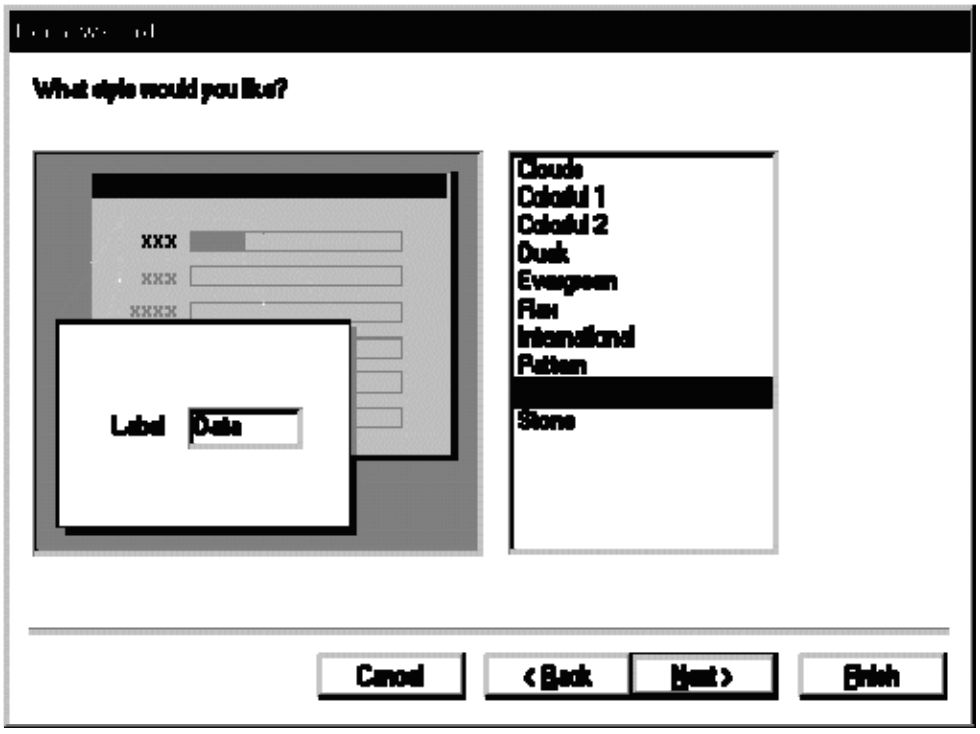


Figure 14.4: Form Wizard Background Dialog box
Source: Microsoft Corporation

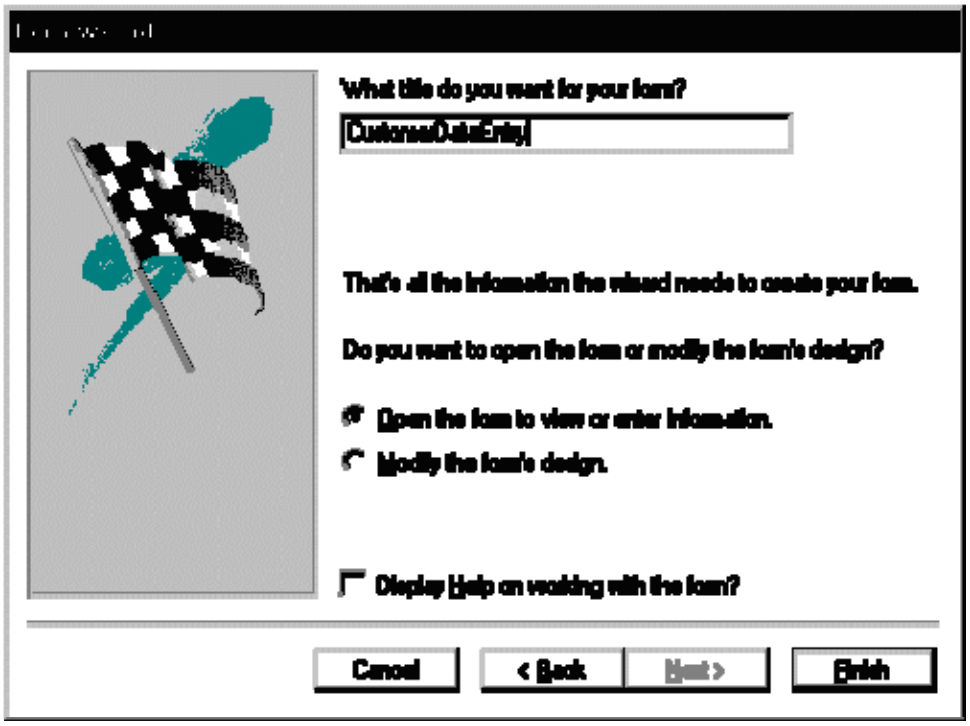


Figure 14.5: Form Wizard Form Title Dialog box
Source: Microsoft Corporation

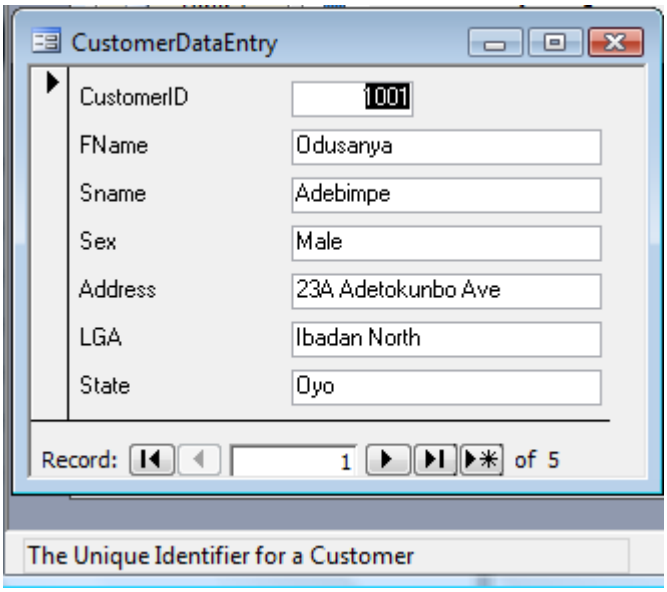


Figure 14.6: Customers Data Entry Form
Source: Microsoft Corporation

3.3 Making Simple Design Changes

In Design View, the structure of the form in terms of its controls and their properties can be manipulated. To make changes to a form:

a. Open the Form.
b. From the View Menu, select Design view.
c. Select any control and then resize it, or move it around to any part of the form.
d. Selected controls can be deleted by pressing CTRL-X
e. The property list of a control can be accessed by doing a right-mouse-click on a selected control and then selecting “Properties” from the menu that appears.
f. Property lists are conveniently organized into categories:
    i. Format --includes properties that affect how the control is displayed
    ii. Data --includes properties that affect where the control gets its data from (notice that some controls get their data from locally defined queries, example “Reports To”)
    iii. Event --includes properties that specify what happens where various events involving the control
g. The properties of a FORM can be accessed by right-clicking on an area that lies outside of the boundaries of the form (“gray area). The most important form properties specify where a form gets its contents from (which table or query), whether it is used for editing or display only, whether it can be resized, etc.
h. New controls can be added to a form by dragging and dropping them from the Toolbox. Newly created controls then need to be sized and moved to their final position in the form. Their properties also need to be set. In most cases, the only

property that matters in the “Control Source”, which specifies the field to which the new control, is bound.

Activity A

Create a data entry form for the Accounts table.

a. Click on the Forms tab on the Access main screen and then click on Create form by using wizard.
b. Select the Accounts table.
c. Select all of the available fields and click on the Next button.
d. Choose a Tabular layout and click on the Next button.
e. Choose the Standard style and click on the Next button.
f. Name the form: AccountsDataEntry
g. Click on the Finish button to create, save and view the new form.
h. Close the form and return to the Access main screen, by pulling down the File menu and choosing Close.

3.4 Creating a Calculated Control

A calculated control is an unbound control that displays totals and other arithmetic computations on a form. You create calculated controls by entering an expression (or formula) to perform the calculation in the control's Control Source property.

To create a calculated control:

1. Display the form in design view.
2. Select the control and click the properties button on the toolbar.
3. Click the data tab and click in the control source box.
4. Type the expression using proper access syntax.

3.5 Form/Subforms

A subform is a form within a form. The primary form is called the main form, and the form within the form is called the subform. Subforms are especially useful when you want to show data from tables or queries with a one-to-many relationship. For example, a Customer form might have a subform that displays each customer's Accounts

The main form and subform are linked so that the subform displays only records that are related to the current record in the main form. For example, when the main form displays a particular customer, the subform displays only accounts for that customer.

Activity B

1. Open the Homework database.

2. Use AutoForm to create and save a columnar form named "Customers," using the Customers table as the underlying data source.
3. Add a text box control with today's date in the bottom-right corner of the Customers form.Hint: You will need to change the text box control's data source to the expression =Today( ).
4. Rearrange the control fields on the form, so that the LastName and FirstName fields appear before the SSN field.
5. Change the Customer form's tab order to reflect the new field order.
6. Delete the DOB field control from the form.
7. Resize the Customers form as necessary, then use the SubForm Wizard to create a subform based on the Insurance Claims table.
8. Modify the Insurance Claims subform so that its Default View property is Single Form View.

Save your changes to the main form and the subform. Then close the form and the Homework database

4.0 Conclusion

A form is nothing more than a graphical representation of a table. You can add, update, and delete records in your table by using a form. A form is very good to use when you have numerous fields in a table. This way you can see all the fields in one screen.

5.0 Summary

In this unit, we have learnt:

civ. Forms are front ends for accessing the data that is stored in database tables or that is generated by queries.
cv. Forms are made up of controls and individual control is typically “bound” to a particular field of the table or query that is associated with the form.
cvi. Properties determine where a form/control gets its data from.
cvii. Forms can be created in two ways: Design view and Wizards
cviii. A subform is a form within a form. The primary form is called the main form, and the form within the form is called the subform.

6.0 Tutor Marked Assignment

1. Which of the following statements about the AutoForm Wizard is NOT true?
    A. The AutoForm Wizard is the fastest and easiest way to create a form in Microsoft Access.
    B. The AutoForm Wizard can only create five types of forms: Datasheet, Columnar, Tabular, PivotTable, or PivotChart.
    C. Forms created with the AutoForm Wizard usually come out looking sharp and professional and don't require any further clean-up work.

   D. The AutoForm Wizard can only create forms based on a single table or query.
2. Which of the following statements is NOT true?
   A. The Field List displays all the fields from a form's underlying table or query.
   B. Click the Field List button on the Toolbar to display the Field List.
   C. You can add fields to a form by dragging them from the Field List onto the form.
   D. The Field List displays all the fields from every table in a database.
3. Controls and their corresponding text labels cannot be moved independently of one another. (True or False?)
4. If you move a control on a form, the Tab Order, in which you advance from one field to the next when you press the Tab key, is automatically updated. (True or False?)
5. A form that has a Datasheet Default View property would display one record at a time in the form. (True or False?)
6. A calculated field... (Select all that apply.)
   A. ...is a bound control.
   B. ...is a control that contains an expression.
   C. ...can perform calculations on fields values, such as =[Cost]*[Commission].
   D. ...can perform calculations on explicit values, such as =2+4.
7. Which of the following set(s) of tables would benefit from a subform? (Select all that apply.)
   A. A Customer table and the Customer Orders table.
   B. A Customer table and Products table.
   C. A Customer table and Foreign Currency table.
   D. A Customer table and a Customer Contacts table.
8. When you add a subform to a main form, Access always recognizes how the two forms are related (True or False?)

7.0 Further Reading and other Resources

Brainbell.com (2008). Microsoft Access Tutorial. Retrieved June 20th, 2008, from http://www.brainbell.com/tutorials/ms-office/Access_2003/
Bcschool.net (2003-2006). Create Database Applications using Microsoft Access, Retrieved June 20th, 2008, from http://www.bcshool.net/staff/accesshelp.htm
Cisnet.baruch.cuny.edu (2009). Microsoft Access Tutorial. Retrieved March 15th, 2009 from http://cisnet.baruch.cuny.edu/holowczak/classes/2200/access/accessall.html.
Databasedev.co.uk (2009). Microsoft Access Tutorial: Retrieved March 15th, 2009 from http://www.databasedev.co.uk/plan-an-access-application.html

Module 3: Design and Development of Database Applications

Unit 4: Introduction to Microsoft Access Reports

1.0 Introduction

A report is an effective way to present your data in a printed format. Because you have control over the size and appearance of everything on a report, you can display the information the way you want to see it.

This unit explains everything you will need in creating and working with reports.

2.0 Objectives

On successful completion you will be able to create and modify a variety of reports.

3.0 Introduction to Reports

Reports present information from tables and queries in a format that looks great when printed. Reports help to print records from tables or queries in a professional way; you can even include calculations, graphics, or a customized header or footer

Reports are similar to queries in that they retrieve data from one or more tables and display the records. However, reports add formatting to the output including fonts, colors, backgrounds and other features. Reports are often printed out on paper rather than just viewed on the screen.

Reports can also summarize and analyze the information in the database. The following are some of the available features in Microsoft Access Reports:

a. Formatting Options: Change the type, size, and color of the fonts used in a report or add lines, boxes, and graphics.
b. Sorting and Grouping Options: Reports are great for summarizing and organizing information.
c. Combine Data from Linked Tables: One report can display data from several related tables or queries.

3.1 Understanding Report Sections

Microsoft Access breaks reports up into separate parts called sections. Each section has its own specific purpose and always prints in the same order on a report. Table 15.1 shows the available sections.

Table 15.1: Report Sections
Source: http://www.brainbell.com/tutorials/ms-office/Access_2003/

Resolution	Description
Report Header	Contains text that appears at the top of the first page of a report, such as the name of the report.
Page Header	Contains text that appears at the top of each page of a report, such as the

Resolution	Description
	report's column headings.
Group Header	Used to place text, such as a group name, at the beginning of each group of records.
Detail	Contains text and the actual fields that are displayed for each record. This would be equivalent to the main body in a word-processing document.
Group Footer	Used to place text and numeric summaries, such as totals or averages, at the end of each group of records.
Page Footer	Contains text that appears at the bottom of each page of a report, such as page numbers.
Report Footer	Contains text that appears at the end of the last page of a report. Often also contains numeric summaries for the report, such as a grand total.

3.2 Creating a Single Report using Wizards

To create a new report:

a. Click on the **Reports** tab in the Main Access menu

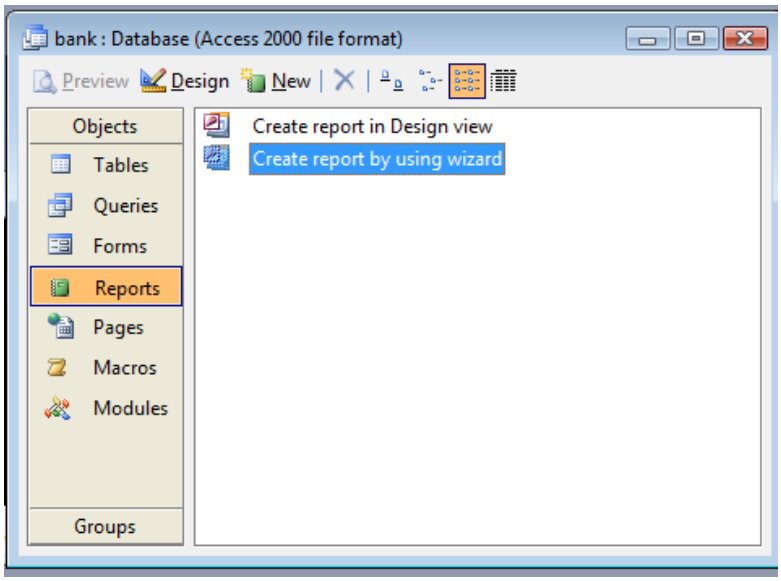


Figure 15.1: Report Screen
Source: Microsoft Corporation

b. Select the **`Create Report by using wizard`** then select the `Customer` table as shown in figure 15.1.
c. Next specify the fields from the Customer table that will appear on the report. In this case, we want all of the fields to appear. Move each of the fields from the

`Available Fields` side over to the `Selected Fields` side as in the following figure as shown in figure 15.2. Then click on the `Next` button.

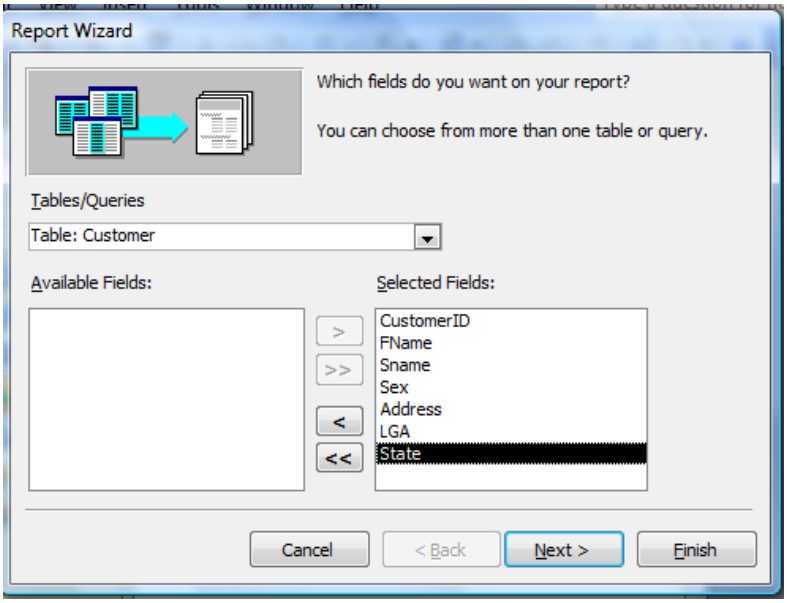


Figure 15.2: Report Field Selection
Source: Microsoft Corporation

d. In the next step, we have the opportunity to add *Grouping Levels* to the report. A grouping level is where several records have the same value for a given field and we only display the value for the first records. In this case, we will not use any grouping levels so simply click on the `Next` button as shown in figure 15.3.

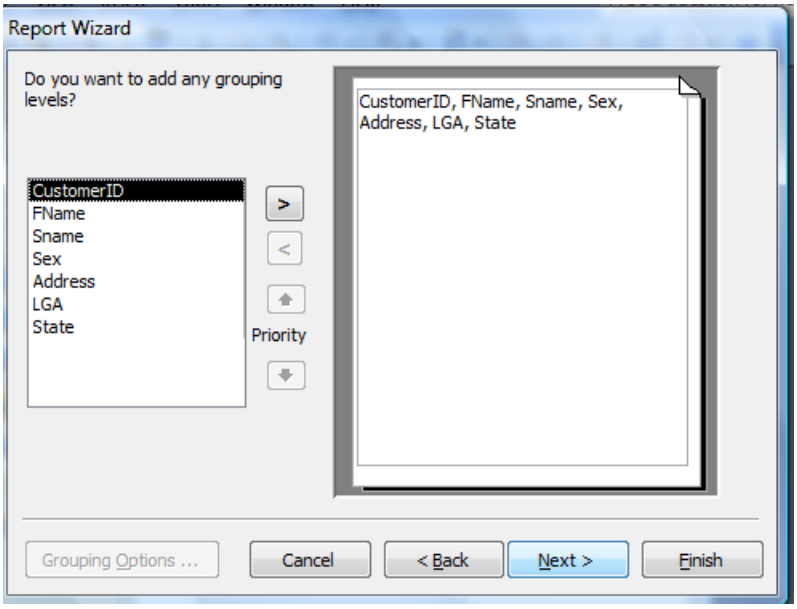


Figure 15.3: Report Grouping Level
Source: Microsoft Corporation

e. In the next step, we are given the opportunity to specify the sorting order of the report. For this example, we will sort the records on the CustomerID field. To achieve this, pull down the list box next to the number `1:` and choose the CustomerID field as shown in the figure 15.4. Then click on the `Next` button.

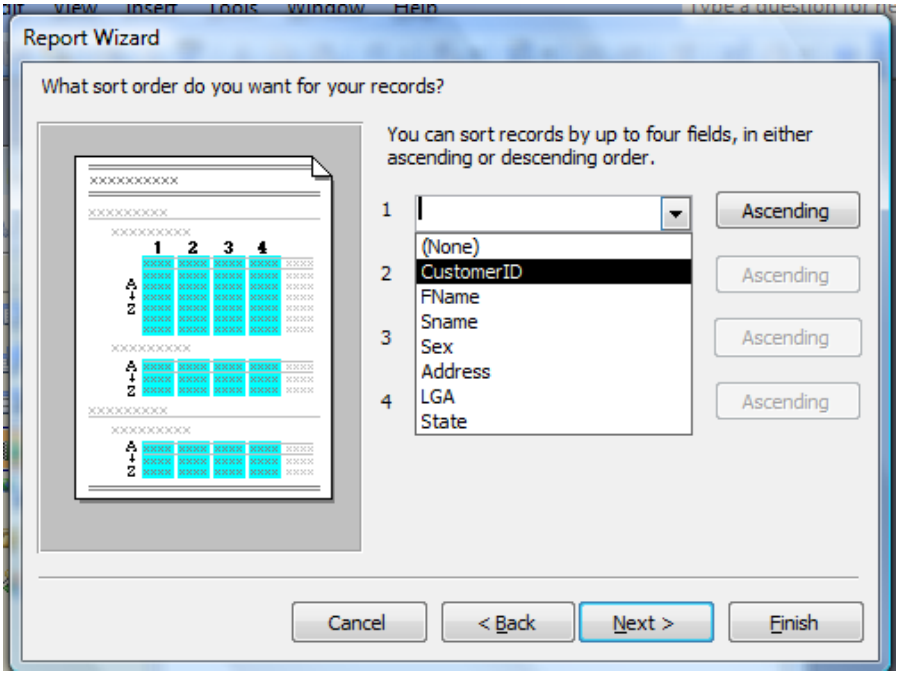


Figure 15.4: Report Sorting Level
Source: Microsoft Corporation

The next step is to specify the layout of the report. The three options are:

i. Columnar - Places the labels to the left of each field. This is similar to a paper form.
ii. Tabular - Places the field labels at the top of the report page and the records are displayed below. This is similar to how a spreadsheet would display the data.
iii. Justified - Places the labels above each field with the fields spread out on the report page.

Generally, reports use the tabular layout. For this example, choose `Tabular` layout and set the page `Orientation` to `Landscape` so that all of the fields will fit across one page. This is shown in the figure 15.5. Click on the `Next` button to continue.

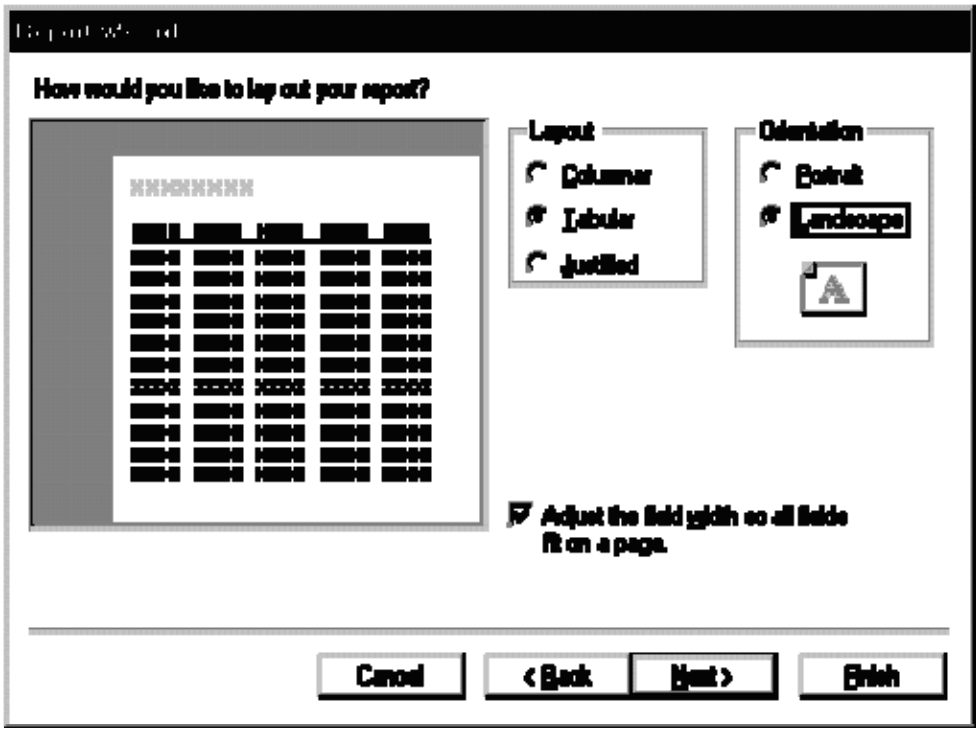


Figure 15.5: Report Tabular Layout
Source: Microsoft Corporation

In the next step, the style of the report can be selected. For this example, choose the `Corporate` style as shown in figure 15.6 and click on the `Next` button to continue.

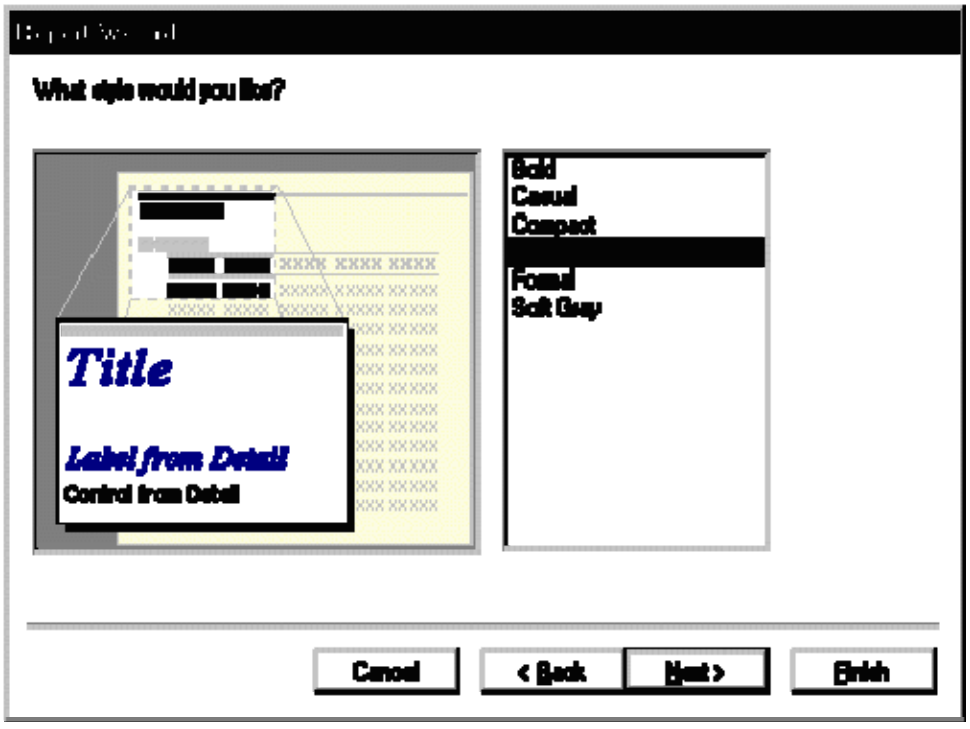


Figure 15.6: Report Style Layout
Source: Microsoft Corporation

Finally, give a name for the new report: `CustomerReport` and then click on the `Finish` button to create, save and display the new report (see figures 15.7 and 15.8).

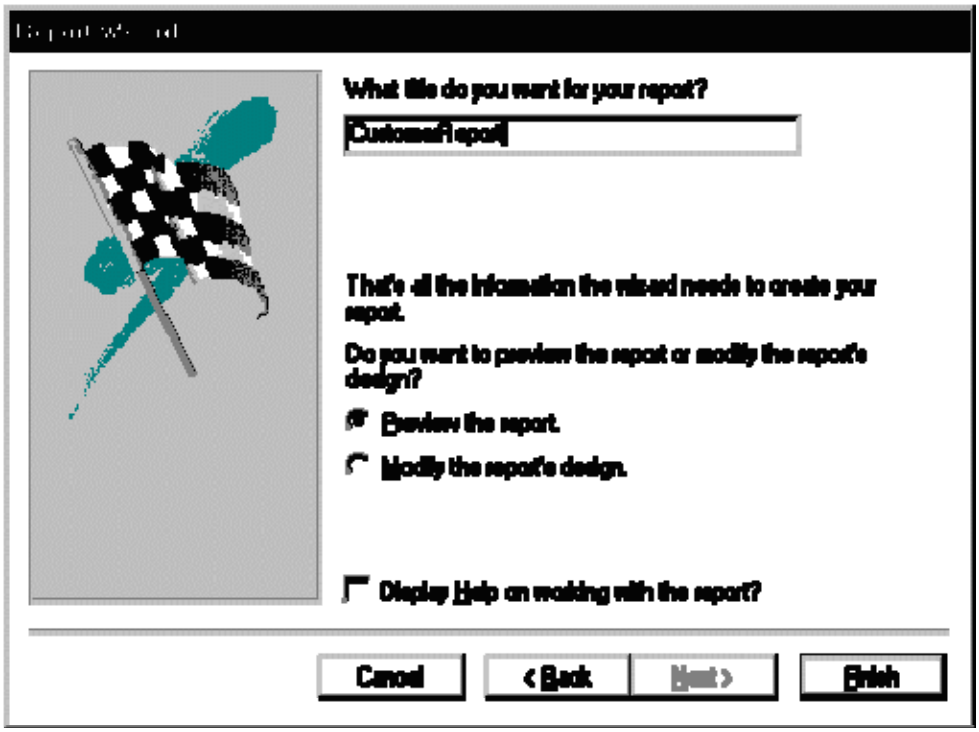


Figure 15.7: Final step in Report Creation with wizard
Source: Microsoft Corporation

The output from the report is shown in the figure below. Note that on some screens, the first or last fields may not display without scrolling over to the left or right.

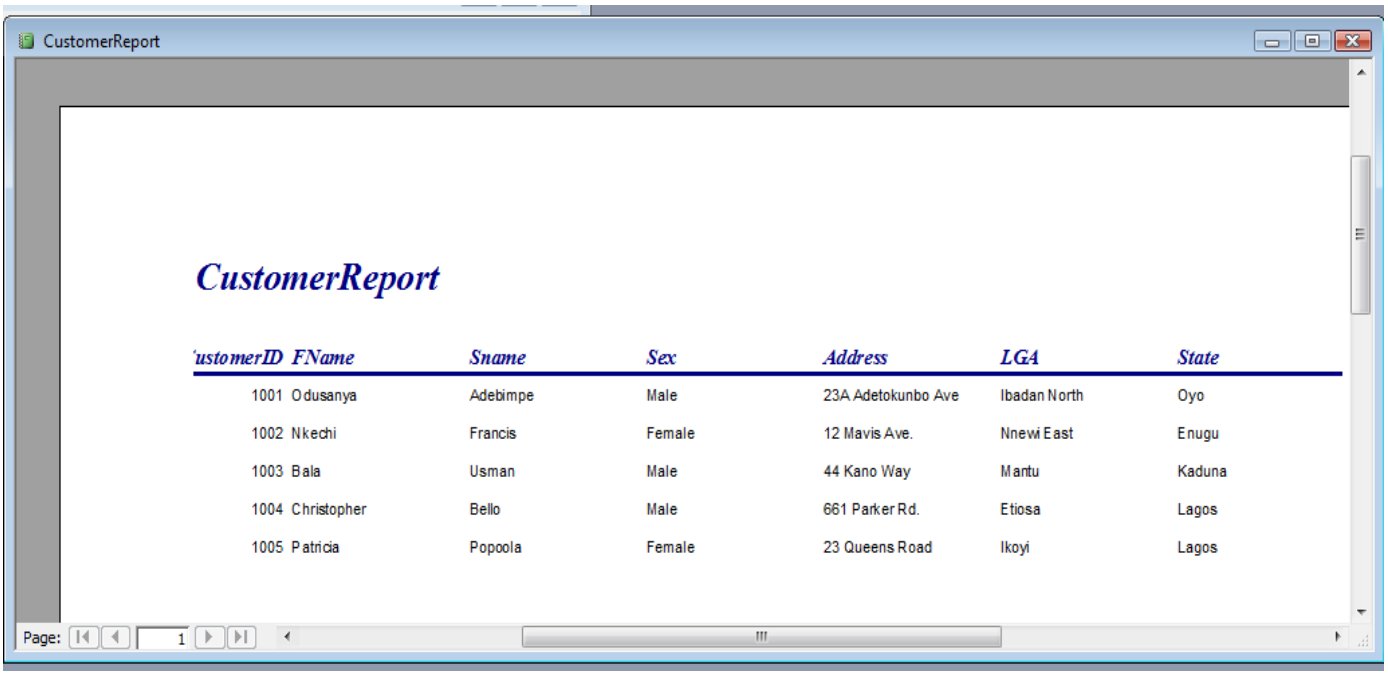
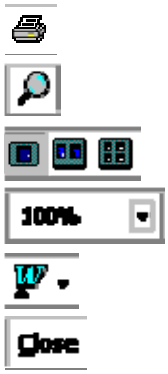


Figure 15.8: Report Sample
Source: Microsoft Corporation

Once the report is displayed, it can be viewed, printed or transferred into Microsoft Word or Microsoft Excel. The button bar across the top of the screen is as shown in figure 15.9



Print the report

Zoom into a region of the report

Display the report as one, two or multiple pages

Zoom into or out of the report

Transfer the report into MS Word

Close the report

Figure 15.9: Report Icons
Source: Microsoft Corporation

To close the report and return to the Access main screen, pull down the `File` menu and choose `Close` or click on the `Close` button.

Activity A

Create a report showing all of the Accounts information.

1. From the Reports tab on the Access main screen, click on the `Create Report using Wizard`.
2. Select the Accounts table.
3. Select all of the fields in the Accounts table by moving them all over to the `Selected Fields` side then click `Next`
4. Group the report by CustomerID by clicking on the CustomerID field and then clicking on the right arrow  button. This is shown in the following figure: Click on the `Next` button.
5. Choose to sort the report on the AccountNumber field. Note that a new button will appear called `Summary Options`. Click on the `Summary Options` button. Choose the Balance field and select the `Sum` option. Choose the option to show both `Detail and Summary` data (see figure 15.10.) Then click on the `OK` button.

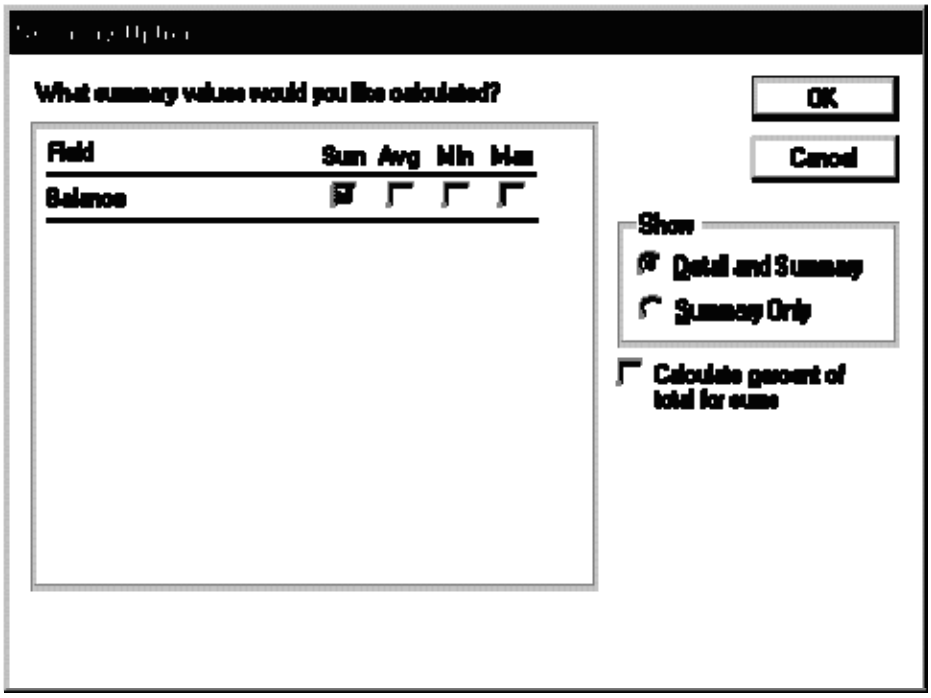
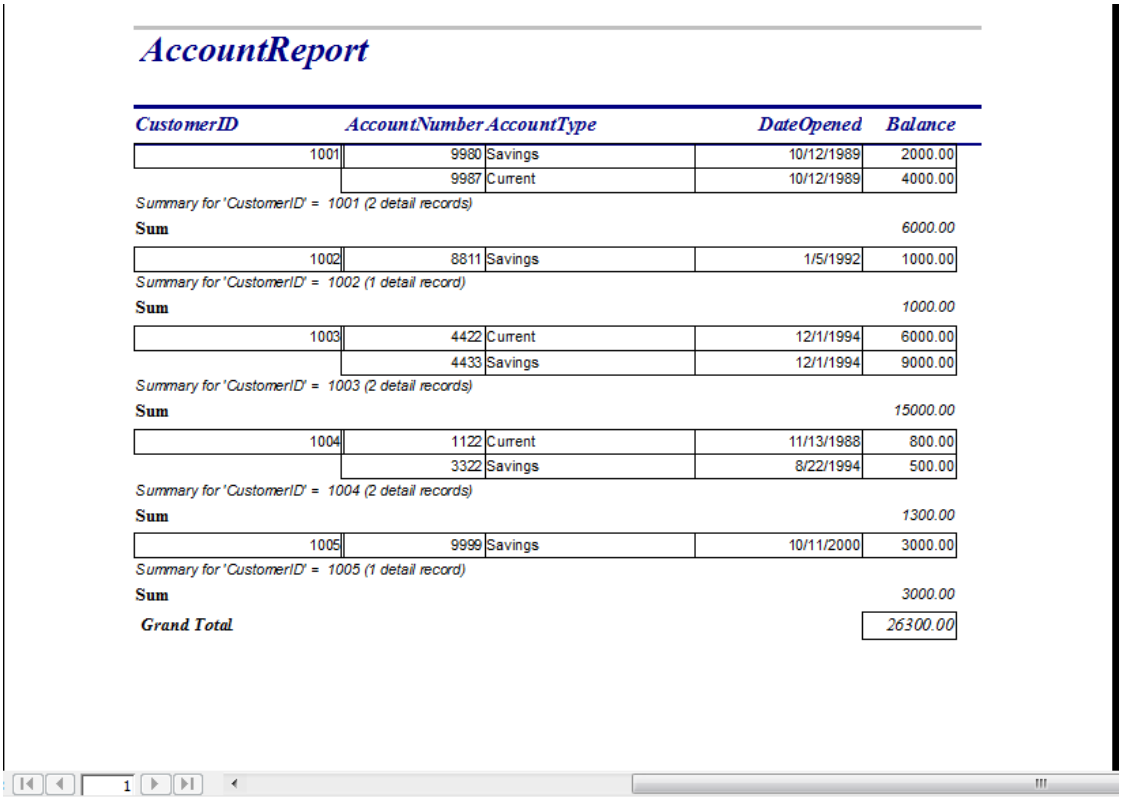


Figure 15.10: Report Summary option
Source: Microsoft Corporation

Click on the `Next` button.

6. Choose a `Block` layout and click on the `Next` button.
7. Choose the Corporate style and the click on the `Next` button.
8. Finally, name the report: `AccountsReport` and click on the `Finish` button to create, save and run the report.

The output from the AccountsReport is shown in Figure 15.11:



AccountReport

CustomerID	AccountNumber	AccountType	DateOpened	Balance
1001	9980	Savings	10/12/1989	2000.00
	9987	Current	10/12/1989	4000.00
Summary for 'CustomerID' = 1001 (2 detail records)				
Sum				*6000.00*
1002	8811	Savings	1/5/1992	1000.00
Summary for 'CustomerID' = 1002 (1 detail record)				
Sum				*1000.00*
1003	4422	Current	12/1/1994	6000.00
	4433	Savings	12/1/1994	9000.00
Summary for 'CustomerID' = 1003 (2 detail records)				
Sum				*15000.00*
1004	1122	Current	11/13/1988	800.00
	3322	Savings	8/22/1994	500.00
Summary for 'CustomerID' = 1004 (2 detail records)				
Sum				*1300.00*
1005	9999	Savings	10/11/2000	3000.00
Summary for 'CustomerID' = 1005 (1 detail record)				
Sum				*3000.00*
Grand Total				*26300.00*

Figure 15.11: Account Report Screen
Source: Microsoft Corporation

To close the report and return to the Access main screen, pull down the `File` menu and choose `Close`.

3.3 Report Controls

Any object that appears on a report is called a control. A text box used to display record information or a column heading are both examples of controls. You add controls to a report by clicking the control you want to use from the tool box and then dragging it onto the report. See Table 15.1 for Toolbox .

Table 15.2: Toolbox Buttons and Controls
Source: http://www.brainbell.com/tutorials/ms-office/Access_2003/

Toolbox Button	Description
	Click this button and then click the control you want to select. To select multiple controls, click this button and hold down the Shift key as you click each control, or drag a rectangle shape around all the controls you want to select.
	Click to use Control Wizards when you add controls to your report.

Toolbox Button	Description
	Creates a text label that appears the same for every record, such as a heading. Most controls already include a text label.
	Creates a text box that displays information from tables and queries in a report.
	Creates a box around a group of option buttons so that the user is only allowed to make one selection from the group box. Normally used in forms, not reports.
	Creates a toggle button. Normally used in forms, not reports.
	Creates an option button (or radio button) that displays data from two or more options. Normally used in forms, not reports.
	Creates a box that is empty or contains a checkmark. Use to display data from a Yes/No field.
	Creates a combo box. Normally used in forms, not reports.
	Creates a list box. Normally used in forms, not reports.
	Creates a button that runs a macro or Visual Basic function. Normally used in forms, not reports.
	Displays a picture by using a graphic file that you specify.
	Inserts an OLE object that is not bound to a field in the current database. Use an Unbound Object Frame to display information from an external source or program, such as a spreadsheet, graphic, or other file.
	Inserts an OLE object that is bound to a field in the database. Use Bound Object Frames to display pictures or other OLE information in the database.
	Inserts a page break.
	Creates a tab control. Normally used in forms, not reports.
	Inserts another report within the main report. Use when you want to show data from a one-to-many relationship.
	Enables you to draw a line in the report.
	Enables you to draw a rectangle in the report.
	Click to display other toolboxes and OLE objects.

3.4 Design View

Design view is used to modify a report so as to make it easier to read and understand. For example, you might want to add or delete a field, change a column heading, or change the locations of the fields in the report. Figure 15.12 shows a sample report in design view.

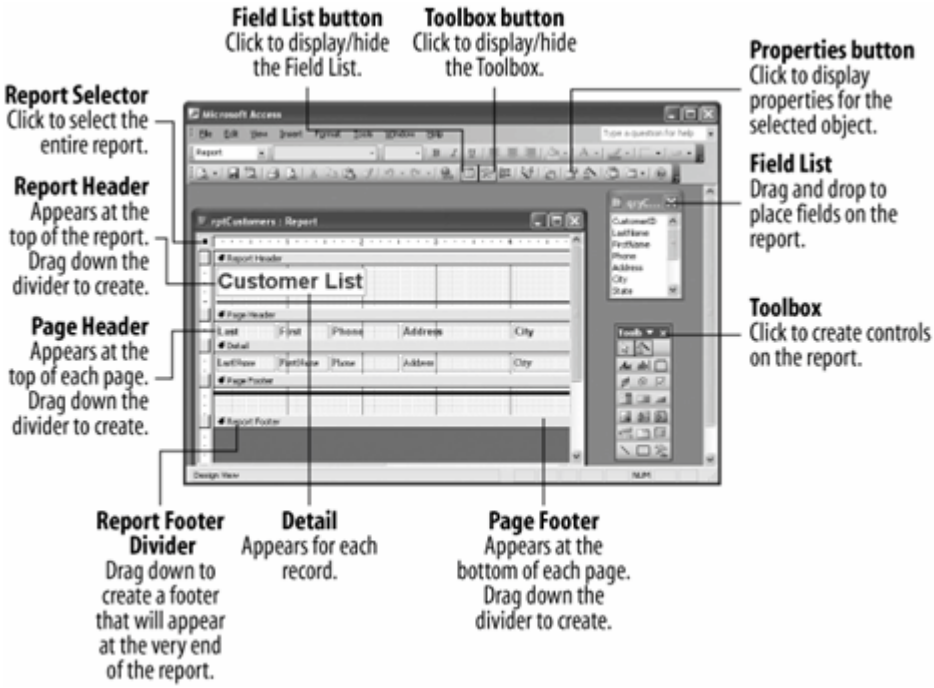


Figure 15.12: A report in Design View
Source: http://www.brainbell.com/tutorials/ms-office/Access_2003/

To modify a report in design view:

a. In the Database window, click the Reports icon in the Objects bar
b. Select the report you want to modify and click the Design button.

Activity B

What are the types of reports that can be created in Microsoft Access?

4.0 Conclusion

Microsoft Access Reports have powerful built in tool that allow you to present your data in a professional way. It is possible to include calculations, graphics, and a customized header or footer in a report.

5.0 Summary

In this unit, we have learnt that:

cix. Reports present information from tables and queries in a format that looks great when printed.
cx. Microsoft Access breaks reports up into separate parts called sections.
cxi. Reports can be created by using report wizard or design view.

cxii. Design view is used to modify a report so as to make it easier to read and understand.

cxiii. Any object that appears on a report is called a control.

6.0 Tutor Marked Assignment

1. Which of the following statements about the AutoReport Wizard is NOT true?
   A. The AutoReport Wizard is the fastest and easiest way to create a report in Microsoft Access.
   B. The AutoReport Wizard can only create two types of reports: Columnar and Tabular.
   C. Reports created with the AutoReport Wizard usually come out looking sharp and professional and don't require further clean-up work.
   D. The AutoReport Wizard can only create reports based on a single table or query .
2. Which of the following statements is NOT true?
   A. The Field List displays all the fields from a report's underlying table or query.
   B. Click the Field List button on the Toolbar to display the Field List.
   C. You can add fields to a report by dragging them from the Field List onto the report.
   D. The Field List displays all the fields from every table in a database.
3. Controls and their corresponding text labels cannot be moved independently of one another. (True or False?)
4. Which of the following statements is NOT true?
   A. You can move a control to a different location on a report by clicking, dragging, and dropping the control.
   B. To add a page number to a report, select View ⟶Header/Footer from the menu and click the Page Number button on the Header/Footer toolbar.
   C. You can resize a report by clicking and dragging the right edge of the report.
   D. You can resize a control by clicking the control to select it, grabbing one of its sizing handles, and dragging and releasing the mouse button when the control reaches the desired size.
5. You want a report to group and total sales by month. Where would you place a calculated control containing the following expression =SUM([Sales]) to calculate the totals for each month?
   A. In the Month Group Footer section.
   B. In the Page Footer section.
   C. In the Report Footer section.
   D. In the Summary section.
6. Which of the following is NOT a report section?
   A. Report Header section.
   B. Page Header section.
   C. Summary section.
   D. Detail section.

7. The only way to sort a report's records is to base the report on a query, which actually does the work of sorting the records. (True or False?)
8. Which of the following expressions is incorrect?
    A. =Total for: [Employee].
    B. =[InvoiceDate]+30.
    C. =[LastName]&" "&[FirstName].
    D. =[Units]*[UnitPrice].
9. You want to track the progress of the stock market on a daily basis. Which type of chart should you use?
    A. Line chart.
    B. Column chart.
    C. Row chart.
    D. Pie chart.
10. How do you adjust a page's margins?
    A. Click and drag the edge of the page to where you want the margin set.
    B. Select Format ⟶Page Setup from the menu, click the Margins tab, and adjust the margins.
    C. Select File ⟶Page Setup from the menu, click the Margins tab, and adjust the margins.
    D. Click the Margins button on the Formatting toolbar.
11. How can you view a report's sorting and grouping options?
    A. Select Format ⟶Sorting and Grouping from the menu.
    B. By double-clicking the Report Selector box in the upper left corner of the report.
    C. Select File ⟶Page Setup from the menu and click the Sorting and Grouping tab.
    D. Click the Sorting and Grouping button on the toolbar.
12. What is the procedure for selecting multiple controls on a report?
    A. Press and hold down the Shift key as you click each object that you want to select.
    B. Use the arrow pointer to draw a box around the object that you want to select.
    C. If the controls are aligned along a horizontal or vertical line, click the horizontal or vertical ruler above or to the left of the controls.
    D. All of these.

7.0 Further Reading and other Resources

Brainbell.com (2008). Microsoft Access Tutorial. Retrieved June 20th, 2008, from http://www.brainbell.com/tutorials/ms-office/Access_2003/
Bcschool.net (2003-2006). Create Database Applications using Microsoft Access, Retrieved June 20th, 2008, from http://www.bcshool.net/staff/accesshelp.htm
Cisnet.baruch.cuny.edu (2009). Microsoft Access Tutorial. Retrieved March 15th, 2009 from http://cisnet.baruch.cuny.edu/holowczak/classes/2200/access/accessall.html.
Databasedev.co.uk (2009). Microsoft Access Tutorial: Retrieved March 15th, 2009 from http://www.databasedev.co.uk/plan-an-access-application.html