**Chapter 1**

**INTRODUCTION TO DATABASE SYSTEMS**

**Objectives:**

At the end of this chapter, the students should be able to:

* Understand the difference between data and information.
* Understand the hierarchy of data from bit to database.
* Differentiate the concepts of database, database systems, and database management systems (DBMS).
* Understand the concept of transaction and its properties.
* Know the different types of DBMS.
* Understand the advantages and disadvantages of using DBMS.
* **DATA vs. INFORMATION**

To understand what drives database design, we must understand the difference between *data* and *information*.

**Data** are raw facts. The word **raw** indicates that the facts have not yet been processed to reveal their meaning. Keep in mind that raw data must be properly formatted for storage, processing, and presentation.

On the other hand**, information** is the result of processing raw data to reveal its meaning in which it requires context. The production of accurate, relevant, and timely information is the key to good decision making.

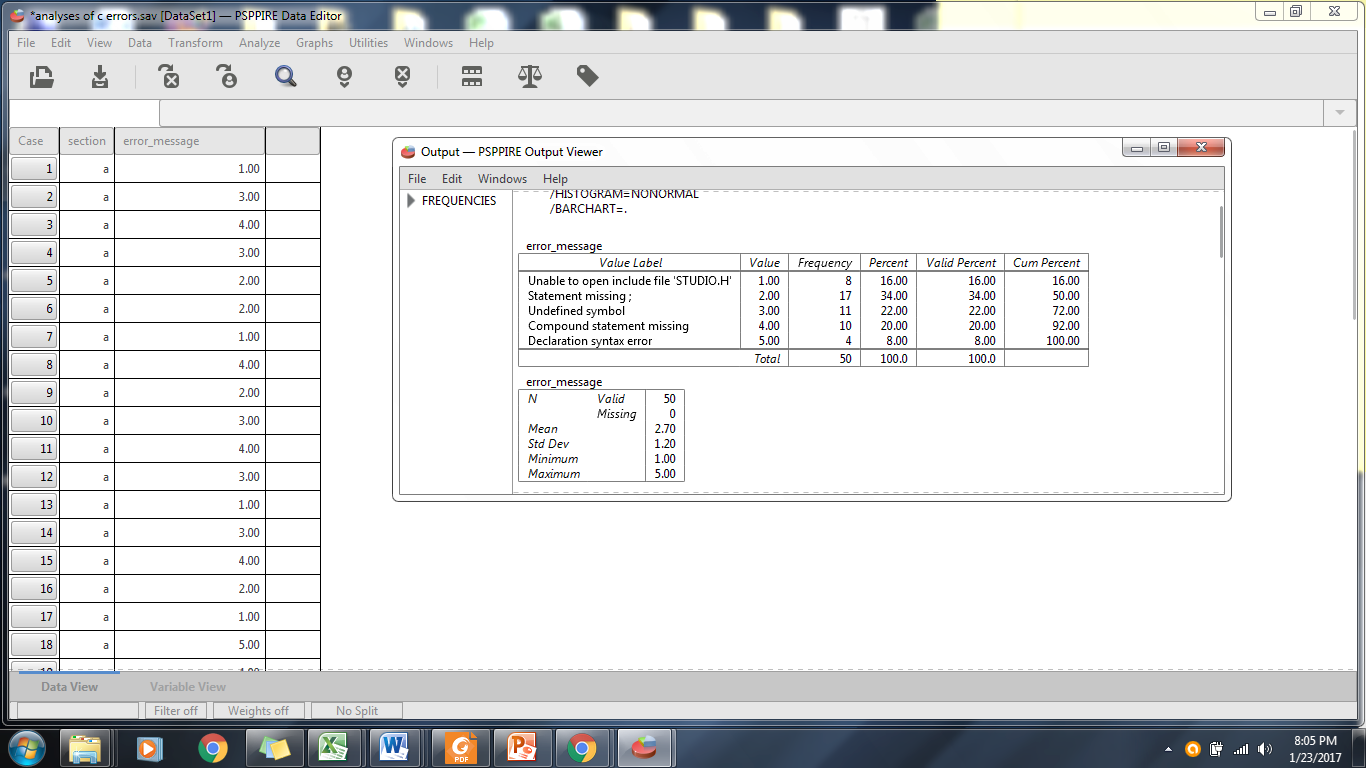
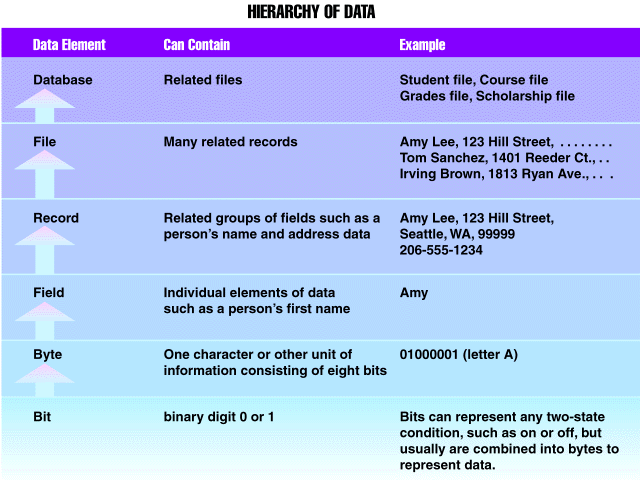


Figure 1. Illustration of transforming raw data into information

* **WHAT IS DATA MANAGEMENT?**

It is a discipline that focuses on the proper generation, storage, and retrieval of data. Given the crucial role that data plays, it is obvious that data management is a core activity for any business, government agency, service organization, or charity.



**Figure 2.** Hierarchy of Data

* **WHAT IS A DATABASE?**

Database is a shared, integrated computer structure that stores a collection of the following:

* + End – user data, which is raw facts of interest to the end user.
  + Description of data characteristics and the set of relationships that link the data found within the database (known as the **metadata**).

The metadata provide a description of the data characteristics and the set of relationships that link the data found within the database. For example, the metadata component stores information such as the name of each data element, the type of values (numeric, dates or text) stored on each data element, whether or not the data element can be left empty, and so on. Therefore, the metadata provide information that complements and expands the value and use of the data.

In short, metadata present a more complete picture of the data in the database. Given the characteristics of metadata, you might hear a database described as a “**collection of *self – describing data****.”*

Database shouldn’t only care about the insertion and modification of the data in the database. At times, it should also focus on how to protect the data stored in the database from unauthorized access.

* **ACID PROPERTIES IN DATABASES**

The terms **A***tomicity*, **C***onsistency*, **I***solation*, and **D***urability* (collectively known as **ACID**), is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, transaction is a single logical operation on the data or an event which occurs on the database.

A transaction typically exhibits two classes of operations; *read* and *write*. The read operation does not change the image of the database in any way. However, a write operation changes the image of the databases through inserting, updating, or deleting data.

To fully understand the properties of transactions, let us try to understand them one by one:

* ***Atomicity***. It requires that each transaction be "*all* or *nothing*“. If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. On the other hand, if one part of the transaction is successful, then it should be reflected in the database.
* ***Consistency****.* The consistency property ensures that any transaction will bring the database from one valid state to another.

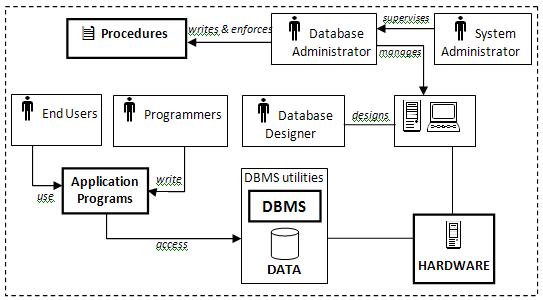
Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.

* ***Isolation***. It ensures that the concurrent execution of transactions result in a system state that would be obtained if transactions were executed serially, one after the other.
* ***Durability****.* This means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.
* **WHAT IS A DATABASE SYSTEM?**

A database system defines and regulates the collection, storage, management, and use of data.

The major parts of a database system are:

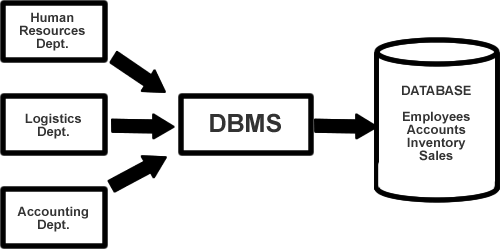
* ***Hardware****.* It is refers to all the database system’s physical devices.
* ***Software****.* It refers to the components that are needed to make the database function fully. There are three software needed for the database system to achieve this goal:
  + ***Operating System****.* It manages all hardware components and makes it possible for all other software to run on the computers. (e.g. Microsoft Windows, Linux, Mac OS)
  + ***DBMS Software****.* It manages the database within the database system. (e.g. Microsoft SQL Server, Oracle, MySQL)
  + ***Application Programs and Utility Software****.* They are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. (e.g. Adobe Dreamweaver, Microsoft Visual Studio)
* ***People****.* This component includes all users of the database system. On the basis of job functions, these users may be describe as the following:
  + ***System Administrators****.* They oversee the database system’s general operations.
  + ***Database Administrators****.* Also known as DBAs, they manage the DBMS and ensure that the database is functioning properly to avoid system glitches in the future.
  + ***Database Designers****.* They simply design the database structure that will dictate the usefulness and effectiveness of the database environment that these people will develop.
  + ***Systems Analysts and Programmers****.* They design and implement the application programs. They also design and create the data – entry screens, reports, and procedures through which end – users access and manipulate the database’s data.
  + ***End – users****.* These are the people who use the application programs to run the organization’s daily operations.
* ***Procedures****.* These are the instructions and rules that govern the design and use of the database system. Procedures are a critical component of the database system. They help to ensure that companies have an organized way to monitor and audit the data that enter the database and the information generated from those data.
* ***Data****.* This refers to the collection of facts stored in the database. Because data are the raw material from which information is gathered, determining what data to enter into the database and how to organize those data is a vital part of the database designer’s job.



**Figure 3.** The Database System Environment

* **WHAT IS A DATABASE MANAGEMENT SYSTEM?**

Commonly known as DBMS, is a collection of programs that manages the database structure and control access to the data stored in the database. For example, imagine that a database as a very well – organized electronic filing cabinet in which powerful software, known as a database management system thathelps manage the cabinet’s contents.

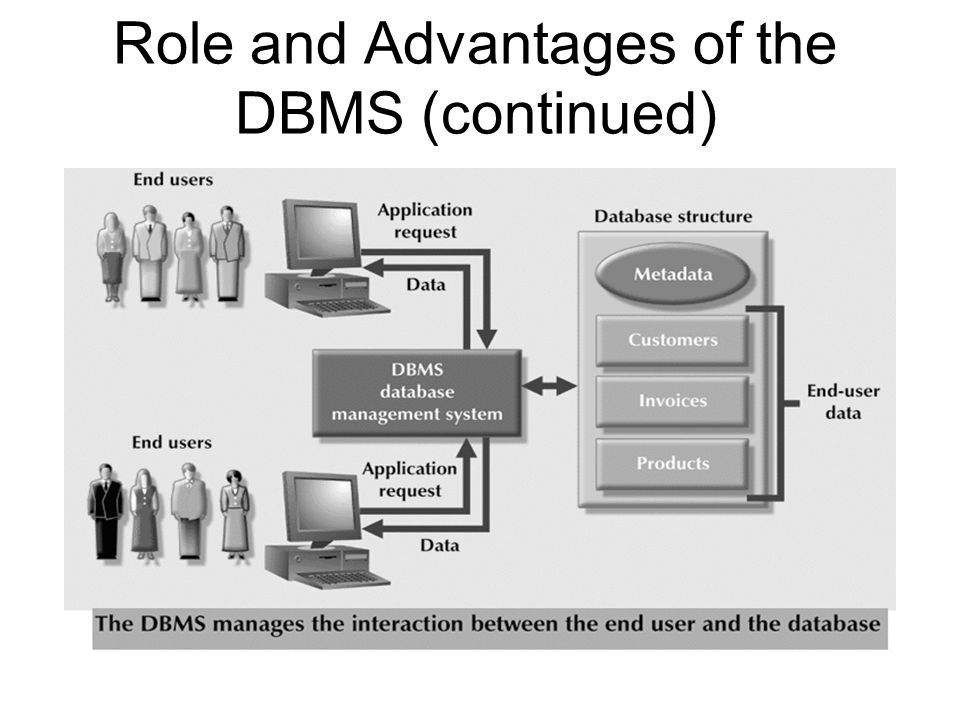


**Figure 4.** Illustration of a DBMS

* **WHAT IS THE ROLE OF A DBMS?**

The DBMS serves as the intermediary between the user and the database. The database structure itself is stored as a collection of files, and the only way to access the data in those files is through the DBMS. The DBMS receives all application requests and translates them into the complex operations required to fulfill those requests.

The DBMS hides much of the database’s internal complexity from the application programs and users.



**Figure 5.** The Role of DBMS

The application program might be written by a programmer using a programming language such as Visual Basic.NET, Java, or C#, or it might be created through a DBMS utility program.

* **WHAT IS THE TYPES OF DBMS?**

The following are types of database management systems that database designers, developers and administrators uses of the years. And behind these types adhere to the different database data models (which will be discussed in Chapter 2):

* *Hierarchical DBMS*
* It is one of the earliest database management systems that was based on the hierarchical model in which data can be organized in the form of free structure or level – by – level manner with one limitation that is ‘*every sub node or child node should have only one parent node*’.
* *Network DBMS*
* Network databases are similar to hierarchical databases by also having a hierarchical structure. However, instead of looking like an upside-down tree, a network database looks more like a **cobweb or interconnected network of records**.  And since more connections can be made between different types of data, network databases are considered more flexible.
* *Relational DBMS*
* Popularly known as RDBMS, Relational databases connect data in different files by using common data elements or a key field. In relational databases, tables or files filled with data are called **relations**, tuples designates a **row** or **record**, and columns are referred to as **attributes** or **fields**.

The relational database has become quite popular for two major reasons. First, relational databases can be used with little or no training. Second, database entries can be modified without redefining the entire structure.

* *Object – Oriented DBMS*
* It is a type of DBMS that stores data elements as objects since it uses the object – oriented programming paradigm.
* **WHAT ARE THE ADVANTAGES OF A DBMS?**

Because data are the crucial raw material from which information is derived, you must have a good method to manage such data. DBMS helps make data management more efficient and effective. In particular, a DBMS provides advantages such as the following:

* *Improved data sharing*. The DBMS helps create an environment in which end users have better access to more data and better managed data. Such access makes it possible for end users to respond quickly to changes in their environment.
* *Improved data security*. The more users access the data, the greater the risks of data security breaches. A DBMS provides a framework for better enforcement of data privacy and security policies.
* *Better data integration*. Wider access to well-managed data promotes an integrated view of the organization’s operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.
* *Minimized data inconsistency*. Data inconsistency exists when different versions of the same data appear in different places. The probability of data inconsistency is greatly reduced in a properly designed database.
* *Improved data access*. The DBMS makes it possible to produce quick answers to ad – hoc queries. From a database perspective, a **query** is a specific request issued to the DBMS for data manipulation. An **ad – hoc query** on the other hand, is a spur-of-the-moment question. The DBMS sends back an answer (called the **query result set)** to the application.
* *Provides Back – up and Recovery*. Centralizing a database provides the schemes such as recovery and backups from the failures which may help the database to recover from the inconsistent state to the state that existed prior to the occurrence of the failure, though methods are very complex.
* *Increased end - user productivity*. The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.
* **WHAT ARE THE DISADVANTAGES OF A DBMS?**

Though the use of a DBMS exhibits many advantages, it has also the following flaws:

* *Database Complexity*. The design of the database system is complex, difficult and is very time consuming task to perform.
* *Substantial Hardware and Software start – up costs*. Huge amount of investment is needed to setup the required hardware and the software needed to run those applications.
* *Damage to database affects virtually all application programs*. If one part of the database is corrupted or damaged because of the hardware or software failure, since we don’t have many versions of the file, al the application programs which are dependent on this database are implicitly affected.
* *Extensive conversion costs in moving form a file – based system to a database system*. Upgrading a file – based system to a database system requires a large amount of cost is incurred in purchasing different tools, adopting different techniques as per the requirement.
* *Initial training required for al programmers and user*. Large amount of human efforts, the time and cost is needed to train the end users and application programmers in order to get used to the database systems.

\*\*\***END OF CHAPTER 1**\*\*\*

**Chapter 2**

**DATA MODELING CONCEPTS**

**Objectives:**

At the end of this chapter, the students should be able to:

* Understand the concept of data modeling and why data models are important
* Know the basic building blocks of data modeling
* Understand the concept of business rules and how does it affect to the design of a database
* Know the different types of data modeling used in database design
* Know the levels of data abstraction
* **WHAT IS A DATA MODEL?**

A **data model** pertains to the organization data elements and standardizes how the data elements relate to one another. Data models describe structured data for storage in data management systems.

A very important aspect about a data model is that when it is implementation – ready, it should contain at least the following components:

* A description of the data structure that will store the end – user data.
* A set of enforceable rules to guarantee the integrity of the data.
* A data manipulation methodology that supports the real – world data transformations.
* **HOW IMPORTANT IS A DATA MODEL?**

Data models can facilitate interaction among the designer, the applications programmer, and the end user. This means that a well – developed data model can even foster improved understanding of the organization for which the database design is developed.

* **BUILDING BLOCKS OF A DATA MODEL**

In its basic sense, a data model has the following:

* *Entity*. It represents a particular type of object in the real world. Because an entity represents a particular type of object, entities are “distinguishable”—that is, each entity occurrence is unique and distinct. (e.g. ***STUDENT*** )
* *Attribute*. It is a characteristics of an entity. Attributes are the equivalent of fields in file systems. (e.g. ***Student Number***, ***Last Name***, ***First Name*** )
* *Relationship.* It describes an association among entities. Data models uses three types of relationships: one – to – one (1:1), one – to – many (1:M), and many – to – many (M:M). (e.g. ***a STUDENT belongs to a SECTION***, ***a STUDENT is enrolled at many SUBJECTS*** )
* *Constraints.* It is a restriction placed on the data. Constraints are important because they help to ensure data integrity. These are normally expressed in the form of rules. (e.g. ***STUDENT passes a SUBJECT if its grade is between 1.00 and 3.00*** , **a *STUDENT NUMBER should be assigned to one and only one STUDENT*** )
* **THE CONCEPT ON BUSINESS RULES**

To properly identify entities, attributes, relationships, and constraints in the database system you are developing, the first step would be always identifying clearly its business rules.

A **business rule** is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. Keep in mind that from a database point of view, *the collection of data becomes meaningful only when it reflects properly defined business rules*.

Examples:

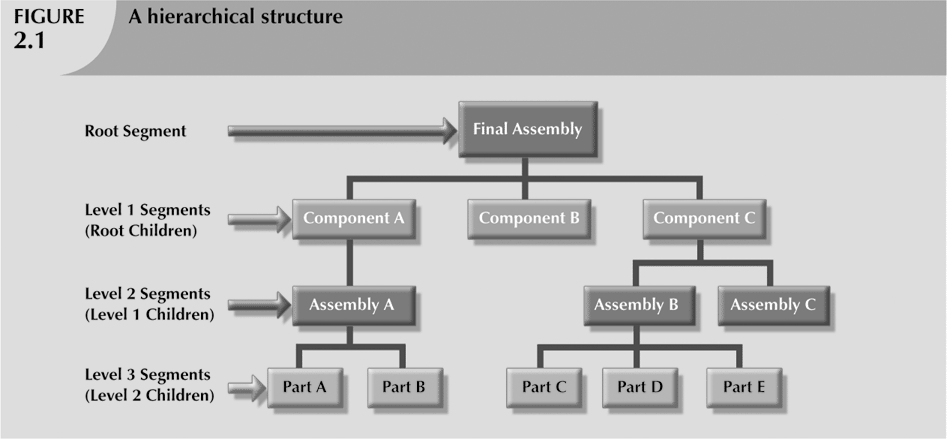
* A **STUDENT** may enroll many **SUBJECTS**.
* A **STUDENT NUMBER** is generated to only one **STUDENT**.
* A **STUDENT** is enrolled in one and only one **COURSE**.

But why business rules are essential to database design? The following reasons may explain the said query:

* To standardize the company’s view of data.
* To constitute communication between users and designers.
* To allow the designer to understand the nature, role, and scope of the data.
* **WHAT ARE THE TYPES OF DATA MODELS?**

The following data models basically represent the “schools of thought” as to what a database is, and what it should do:

* *Hierarchical Model.* Developed in the 1960s, its basic logical structure is represented by an upside – down “tree”. The hierarchical structure contains levels, or segments. It basically depicts a set of one-to-many (1:M) relationships between a parent and its children segments in which each parent can have many children and each child has only one parent.

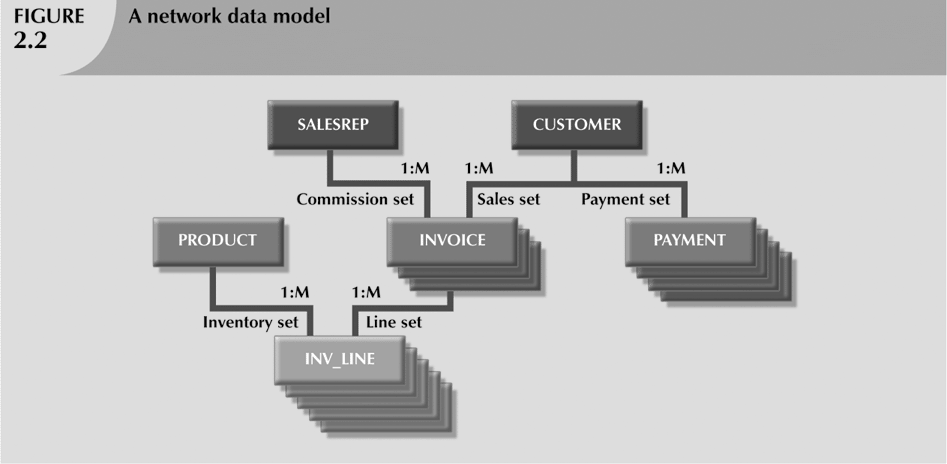


**Figure 1.** The Hierarchical Data Model Structure

* *Network Model.* Unlike the hierarchical model, the network model allows a record to have more than one parent. It was created based from the hierarchical model due to the following reasons:
  + To represent complex data relationships more effectively.
  + To improve database performance.
  + To impose a database standard.

While the network database model is generally not used today, the definitions of standard database concepts that emerged with the network model are still used by modern data models. And these are:

* *Schema.* Conceptual organization of entire database as viewed by the database administrator.
* *Subschema.* Defines database portion “seen” by the application programs that actually produce the desired information from data contained within the database
* *Data Management Language (DML).* It defines the environment in which data can be managed.
* *Data Definition Language (DDL).* It enables database administrator to define schema components.



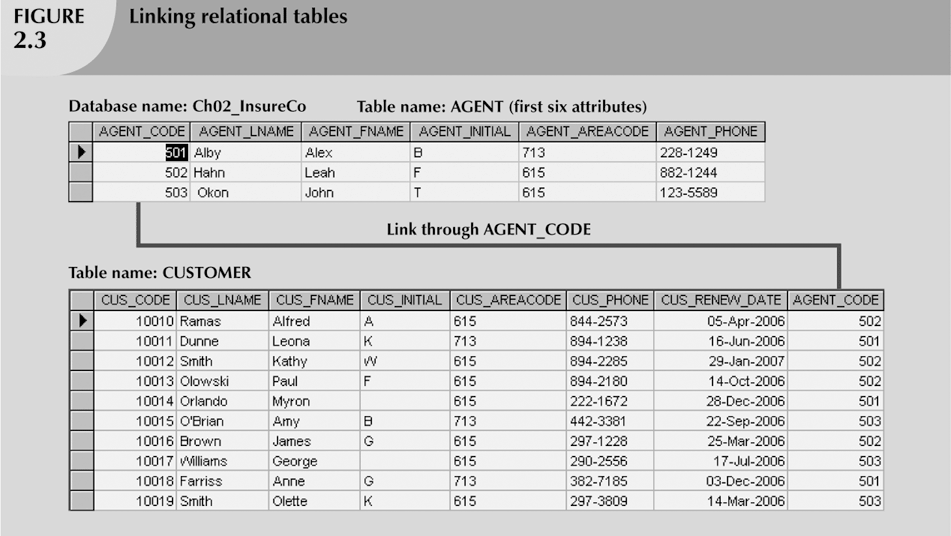
**Figure 2.** A Network Model

* *Relational Model.* Introduced by E. F. Codd on 1970, the relational model foundation is a mathematical concept known as a relation. To avoid the complexity of abstract mathematical theory, you can think of a **relation** (sometimes called a **table**) as a matrix composed of intersecting rows and columns.

Each row in a relation is called a **tuple**. Each column represents an **attribute**. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts.

The relational data model is implemented through a very sophisticated **relational database management system (RDBMS).** In addition to a host of other functions that make the relational data model easier to understand and implement.

The rise to dominance of the relational data model is due in part to its powerful and flexible query language such as the Structured Query Language or SQL. **Structured Query Language (SQL)** allows the user to specify what must be done without specifying how it must be done.



**Figure 3.** An Implementation of the Relational Model

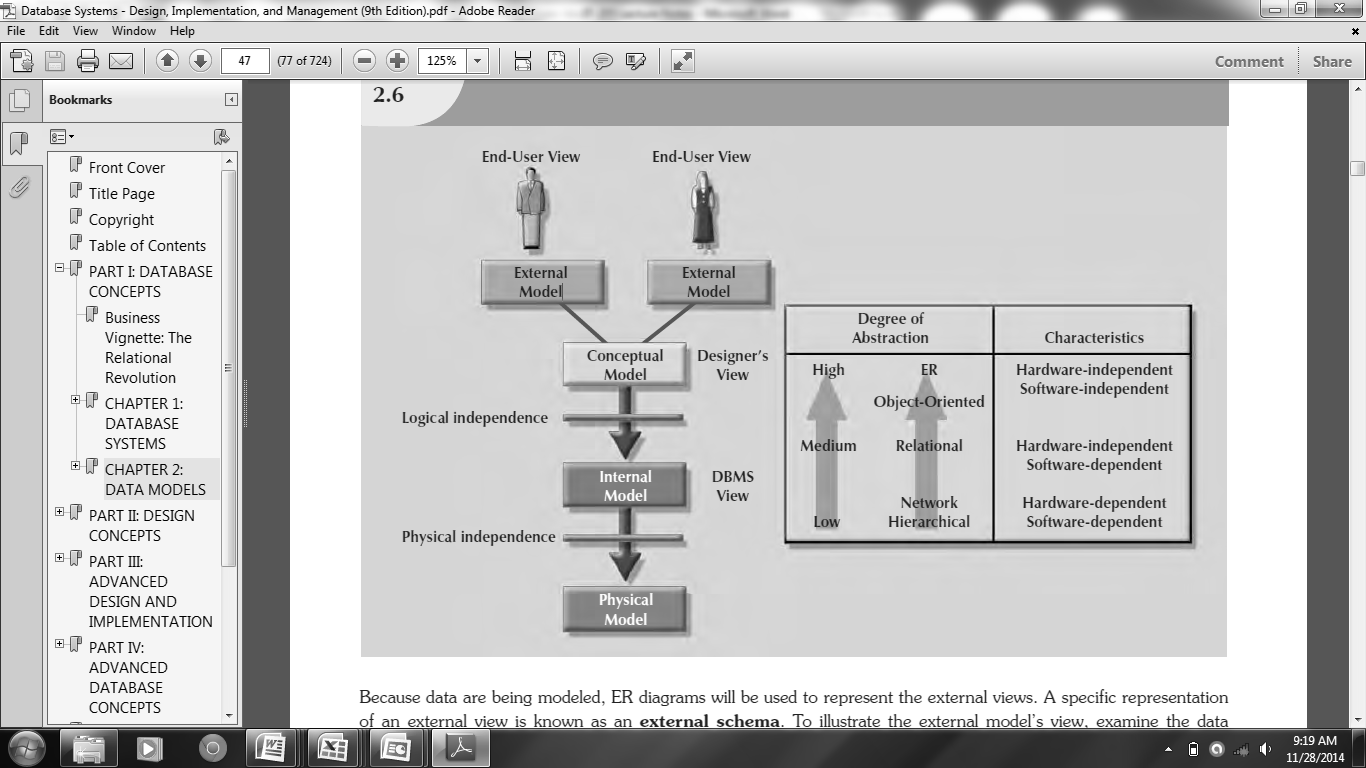
* *Object – Oriented Model.* **Object-oriented data model (OODM)** is the basis for the object-oriented database management system (OODBMS). OODM is said to be a semantic data model in which it describes the meaning of its instances.

Object described by its factual content like the relational model’s entity. It includes information about relationships between facts within object, and relationships with other objects unlike the relational model’s entity. Object becomes basic building block for autonomous structures.

 **Figure 3.** An Ilustration of the OO Data Model

* **WHAT ARE THE LEVELS OF DATA ABSTRACTION?**

In the early 1970s, the **American National Standards Institute** (**ANSI**) Standards Planning and Requirements Committee (SPARC) defined a framework for data modeling based on degrees of data abstraction. The ANSI/SPARC architecture (as it is often referred to) defines three levels of data abstraction: external, conceptual, and internal.



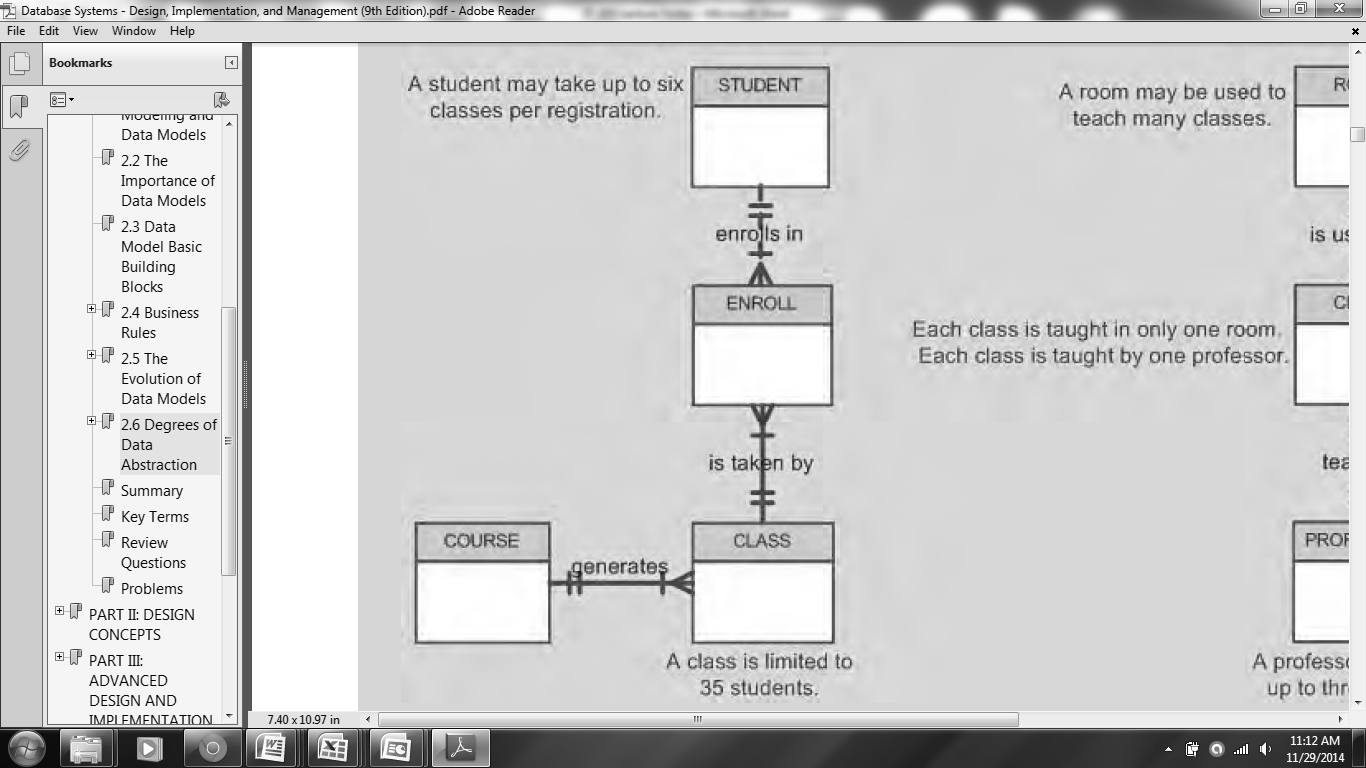
**Figure 4.** Data Abstraction Levels

* *External Model.* The **external model** is the end users’ view of the data environment. The term ***end users***refer to people who use the application programs to manipulate the data and generate information.

End users usually operate in an environment in which an application has a specific business unit focus. Companies are generally divided into several business units, such as sales, finance, and marketing.

Each business unit is subject to specific constraints and requirements, and each one uses a data subset of the overall data in the organization. Therefore, end users working within those business units view their data subsets as separate from or external to other units within the organization. Because data are being modeled, ER diagrams will be used to represent the external views.

A specific representation of an external view is known as an **external schema**

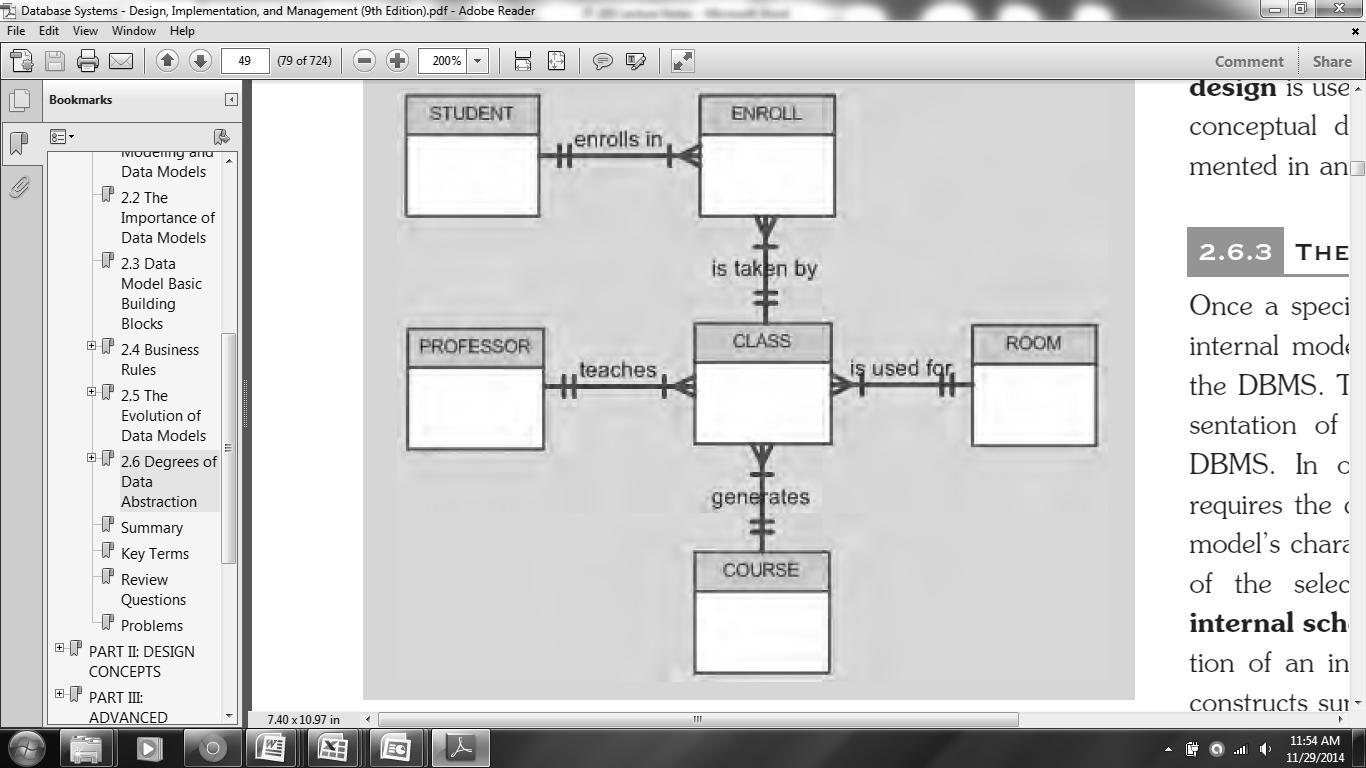


**Figure 5.** A sample external view for a student registration data model

* *Conceptual Model.* The **conceptual model** represents a global view of the entire database as viewed by the entire organization. That is, the conceptual model integrates all external views such as entities, relationships, constraints, and processes into a single global view of the data in the enterprise. Having identified the external views, a conceptual model is used, graphically represented by an entity – relationship diagram (ERD), to integrate all external views into a single view.

Also known as a **conceptual schema**, it is the basis for the identification and high-level description of the main data objects, avoiding any database model –specific details.

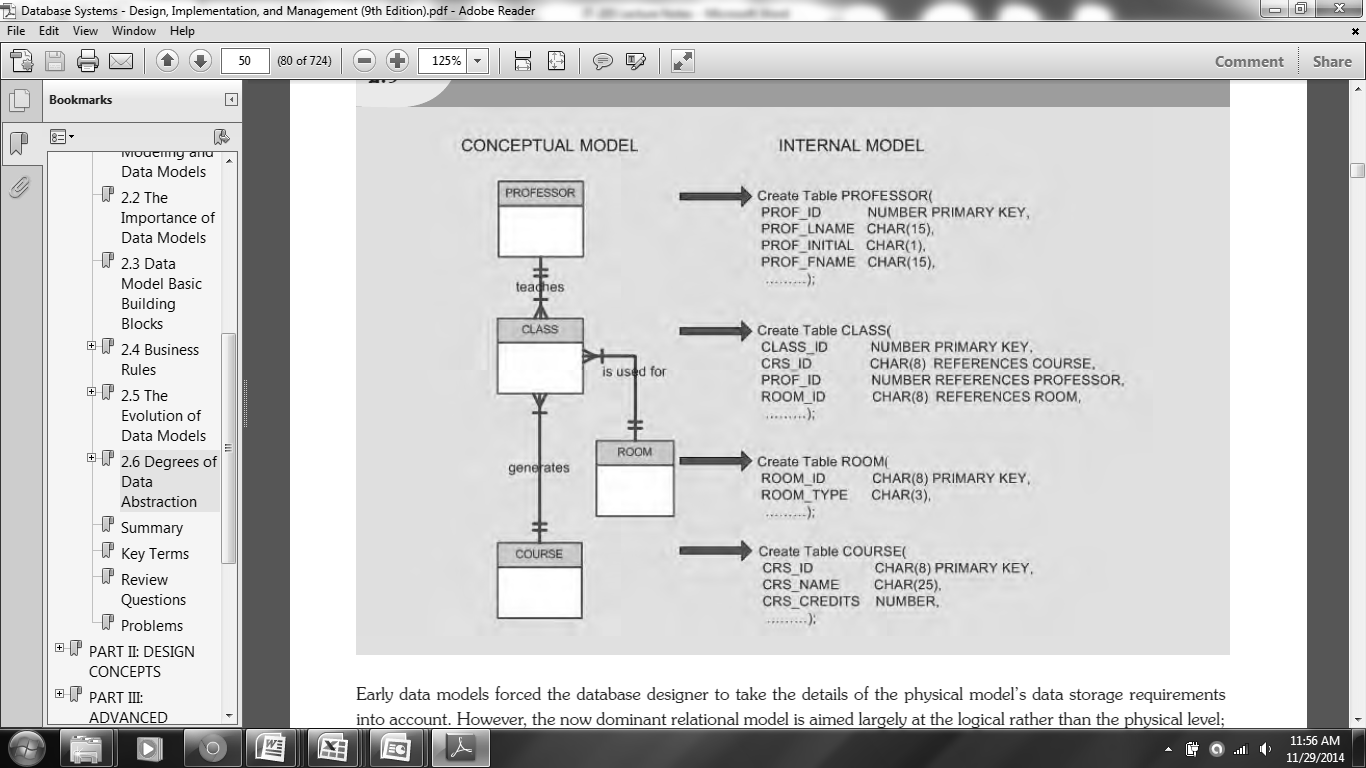
The most widely used conceptual model is the ER model. Remember that the ER model is illustrated with the help of the ERD, which is, in effect, the basic database blueprint. The ERD is used to graphically representthe conceptual schema.



**Figure 6.** The Conceptual View of an Enrollment System

* *Internal Model.* Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The **internal model** is the representation of the database as “seen” by the DBMS. In other words, the internal model requires the designer to match the conceptual model’s characteristics and constraints to those of the selected implementation model.

An **internal schema** depicts a specific representation of an internal model, using the database constructs supported by the chosen database.



**Figure 7.** Conversion from the Conceptual Model to Internal Model

* *Physical Model.* The **physical model** operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices, making it both software and hardware dependent.

The storage structures used are dependent on the software such as the DBMS and the operating system and on the type of storage devices that the computer can handle. The precision required in the physical model’s definition demands that database designers who work at this level have a detailed knowledge of the hardware and software used to implement the database design.

\*\*\***END OF CHAPTER 2**\*\*\*

**Chapter 3**

**THE RELATIONAL DATABASE MODEL**

**Objectives:**

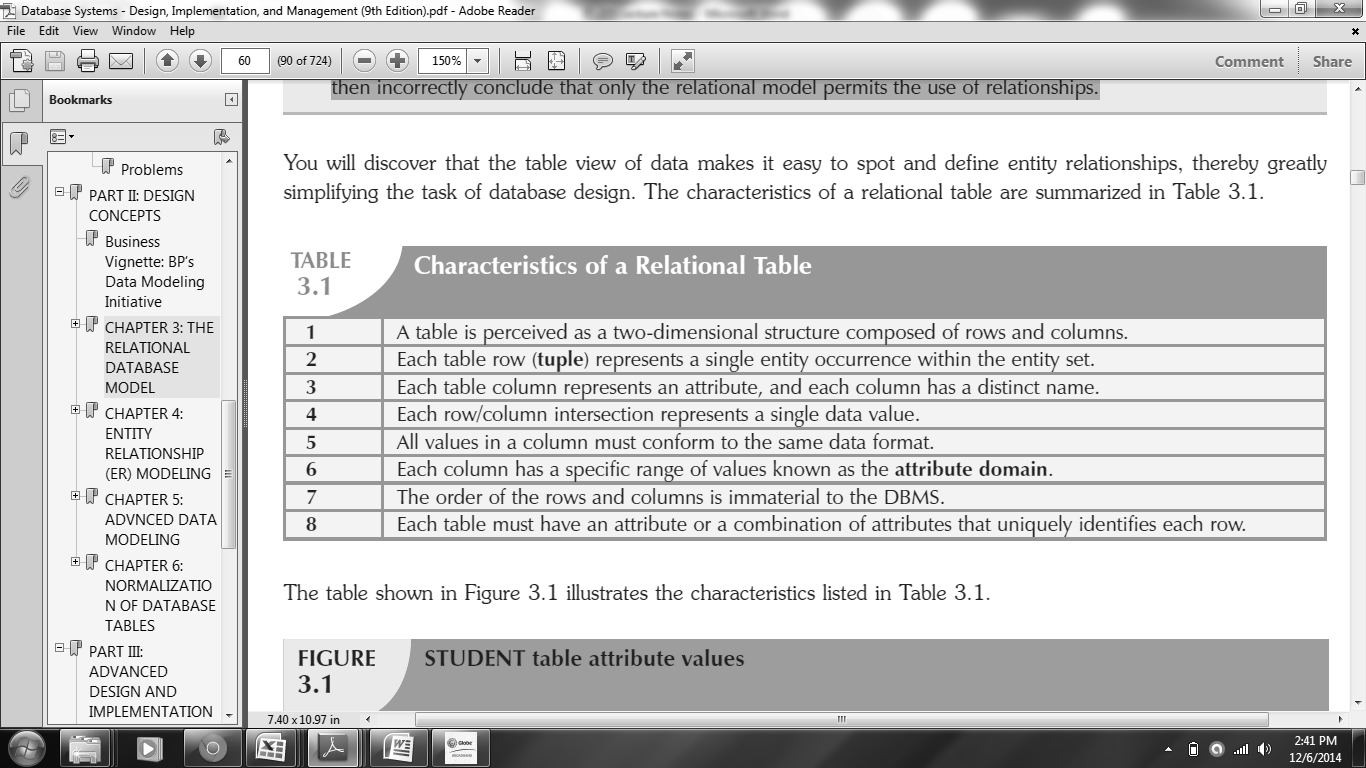
At the end of this chapter, the students should be able to:

* Understand the concept of relations as the relational model’s basic component.
* Understand the characteristics of a relational table.
* Know the different data types applicable to the relational database model.
* Know the concept keys in a relational database model and how each keys may differ from one another.
* Know how develop an Entity Relationship Diagram for database modeling.
* **THE LOGICAL VIEW OF THE RELATIONAL DATABASE MODEL**

The idea of the relational database model is that data in a specific table is directly associated with all other items in that same table. The logical view of the relational database model is facilitated by the creation of data relationships based on a logical construct known as a **relation**. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table.

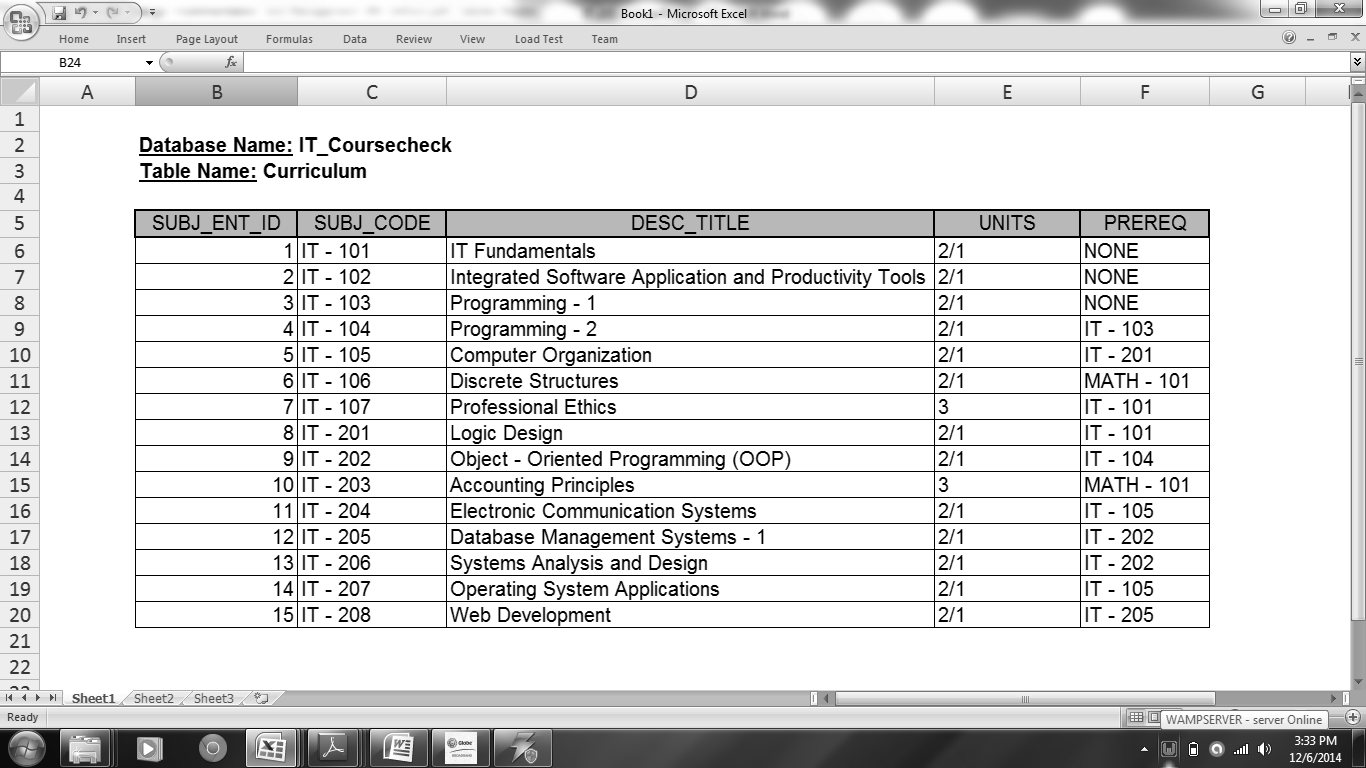
A **table** is perceived as a two – dimensional structure composed of rows and columns. A table is also called a **relation**because the relational model’s creator, E. F. Codd, used the term **relation**as a synonym for **table**. It was based on the mathematical set theory from which Codd derived his model. Because the relational model uses attribute values to establish relationships among tables, many database users incorrectly assume that the term relationrefers to such relationships. Many then incorrectly conclude that only the relational model permits the use of relationships.

Table 1 shows the characteristics of a table in lieu of the relational database model.



**Table 1.** Characteristics of Relational Table

Figure 8 shows the illustration a relational table’s characteristics as listed in Table 1.



**Figure 8.** Curriculum Table Attribute Values

* **THE CONCEPT OF DATA TYPES IN A RELATIONAL DATABASE MODEL**

A data type comprises the forms data can take inside the database. There are many different types of data types, which vary often more in name than anything else with respect to different to DBMS software.

They can be divided into three separate categories:

* *Simple Data Types.* This includes basic validation and formatting requirements placed on to individual values (this will also discussed in details on Chapter 4). Simple data types may also be sub – categorized into the following:
  + *Strings.* A **string** is a sequence of one or more characters. Strings can be **fixed – length strings** in which it will always store the specified length declared for that data type. They can be also **variable – fixed length** strings in which it allows storage into a data type as the actual length of string, as long as a maximum limit is not exceeded.

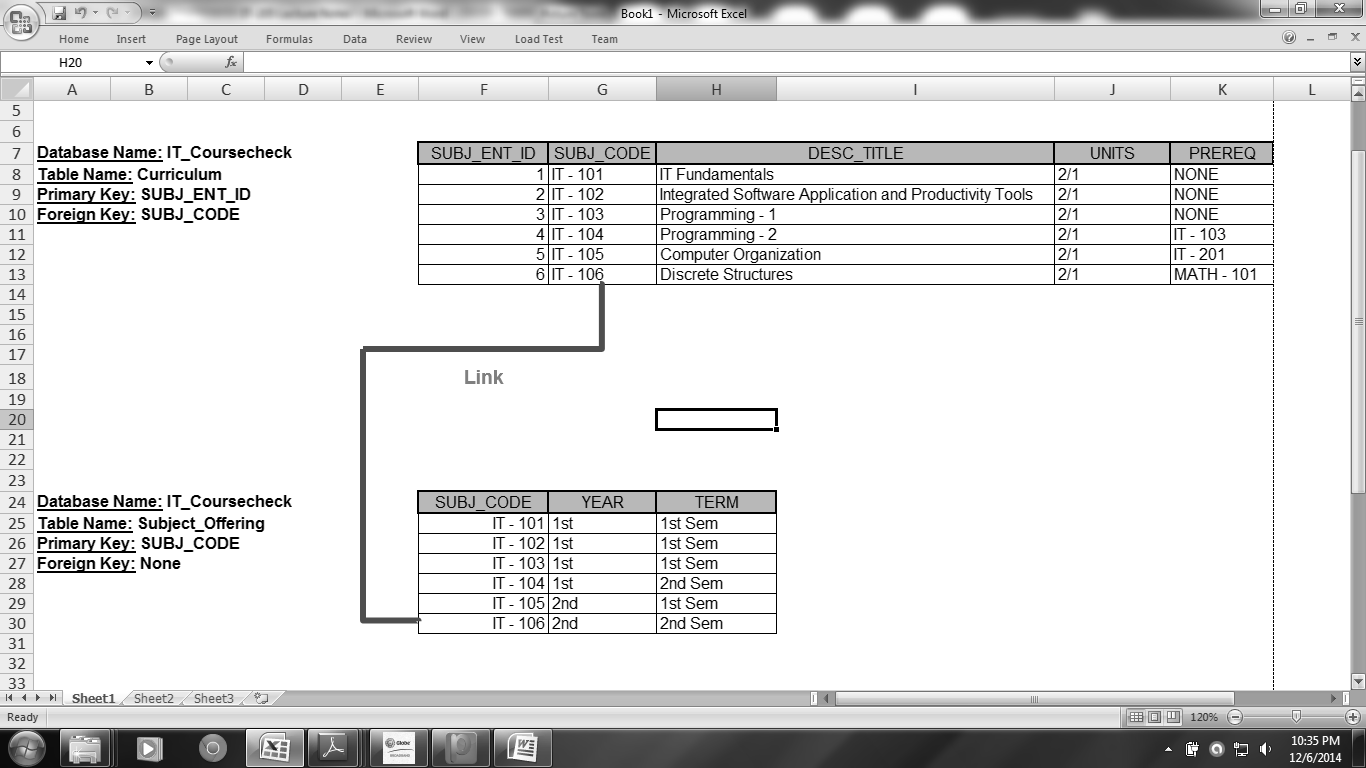
* + *Numbers.* Numeric data types are often the most numerous field data types in many database tables. Here are some of them:
    - Integers
    - Fixed – length decimals
    - Floating points
    - Dates and times
* *Complex Data Types.* This encompasses object data types. Available object data types vary for different relational databases. Some relational databases provide more object – relational attributes and functionality than others. Complex data types include any data types breaching the object – relational database divide including items such as binary objects, reference pointers, collection arrays and even the capacity to create user – defined types.
* *Specialized Data Types.* These are present in more advanced relational databases catering to inherently structured data such as XML documents, spatial data, multimedia objects and even dynamically definable data types.
* **WHAT IS A KEY?**

A **key** consists of one or more attributes that determine other attributes in a relational table. In relational databases, keys are important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. Therefore, a proper understanding of the concept and use of keys in the relational model is very important.

* **WHAT ARE THE TYPE OF KEYS COMMONLY USED IN A RELATIONAL DATABASE?**

The following keys are often found on relational tables in which they play important roles to the functionality on the whole relational database:

* *Primary Key.* It is used to uniquely identify a record in a table. Unique identification for each record is critical because there is no other way to find a record without the possibility of finding more than one record, if the said unique identifier is not used. A primary key can be also used to define relationships between tables.
* *Candidate Keys.* Sometimes called as a unique key, it is created on a field containing only unique values throughout the entire table similar to a primary key. However, candidate keys are not used to define relationships between tables compare to primary keys. They only ensure uniqueness across a table.
* *Foreign Keys.* They are copies of primary keys created into child tables to form the opposite side of the link in an inter – table relationship in which it establishes a “relational database relation”. A foreign key defines the reference for each record in the child table, referencing back to the primary key in the parent table.



**Figure 9.** Illustration on the Use of Primary and Foreign Keys

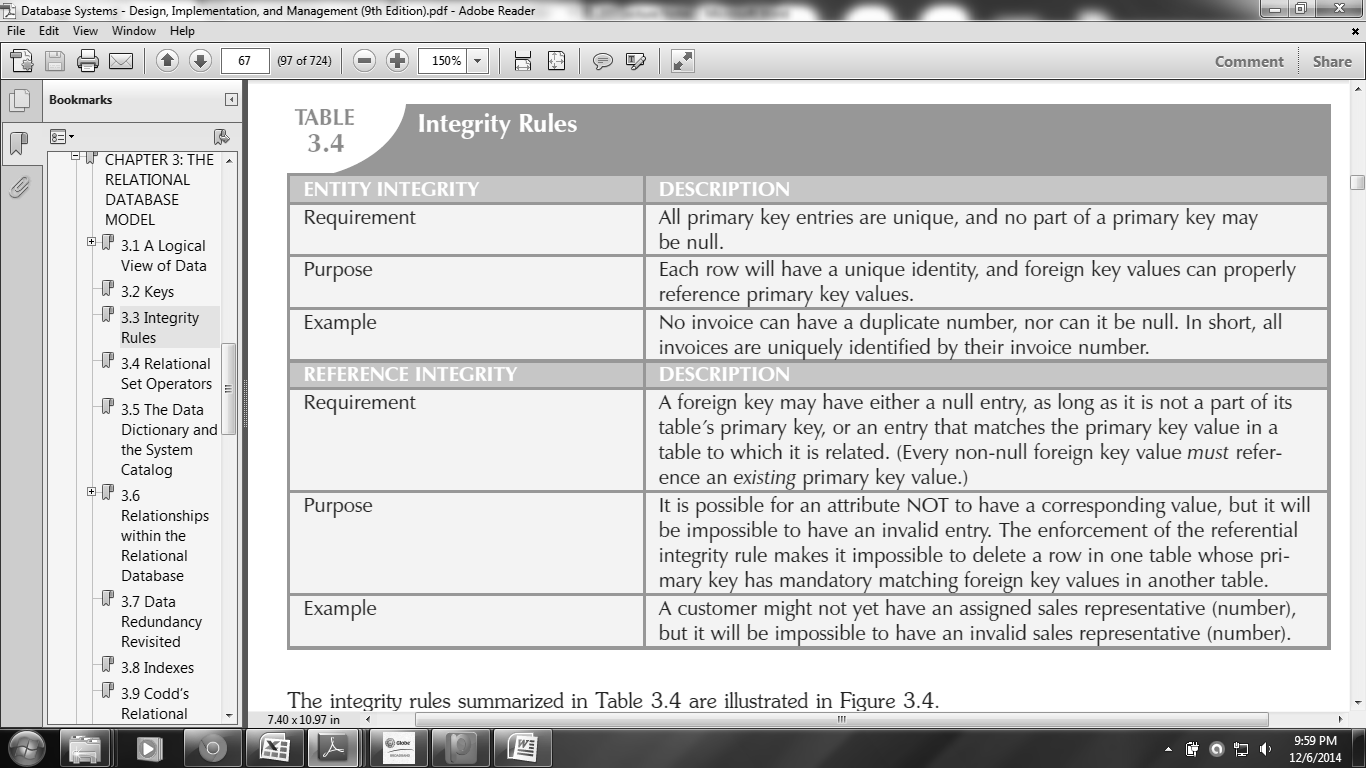
* **WHAT IS DATA INTEGRITY?**

In a relational database, **data integrity** refers to a condition in which the data in the database is in compliance with all entity and referential integrity constraints. The integrity of data is the validity of data, wherein possible compromises to data integrity include human error at data entry, network transmission errors, software bugs and virus infections, hardware malfunction, disk errors, and natural disasters.

To counter these compromises to data integrity, the most significant prevention mechanisms are regular database backups, computer security in all forms, and a properly designed interfaces restricting how data is entered by data users.

* **INTEGRITY RULES**

Relational database integrity rules are very important to good database design. Many (but by no means all) RDBMSs enforce integrity rules automatically. However, it is much safer to make sure that your application design conforms to the entity and referential integrity rules as shown in Table 2.



**Table 2.** Entity and Referential Integrity Rules

From Figure 8, the following features can be established based from the entity and referential integrity rules from Table 2:

* In terms of *entity integrity*, the **Curriculum** table’s primary key is **SUBJ\_ENT\_ID**. Notice that the Curriculum table’s primary key column has **no null entries** and **all entries are all unique**. Similarly, the **Subject\_Offering** table’s primary key is **SUBJ\_CODE**, and its column is all free of null entries.
* In terms of *referential integrity*, the **Curriculum** table *contains a foreign key*, **SUBJ\_CODE** that links entries in the **Curriculum** table to the **Subject\_Offering** table.

Basically, primary and foreign keys automatically verify against each other. Primary and foreign key references are the connections establishing and enforcing referential integrity between tables.

On the other hand, a table’s primary key should never have null values or duplicate entries. That is because each row in the table should have a unique identity that will be based from the table’s primary key.

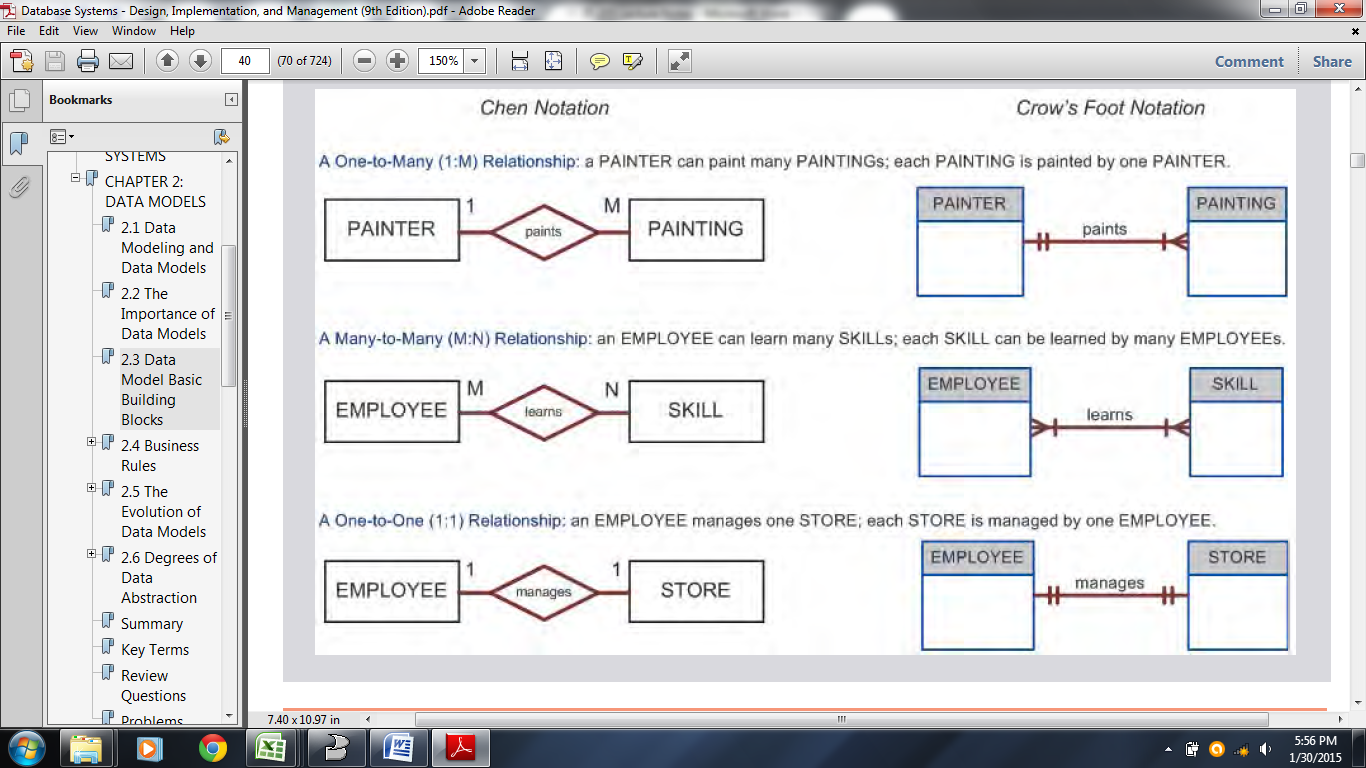
* **REPRESENTING RELATIONSHIPS USING AN ERD**

In a relational database model, there are three relationship classifications that is applicable depending on the actual database design and implementation:

* The **1:M** (or one – to – many) relationship is the relational modeling ideal. Therefore, this relationship type should be the standard in any relational database design.
* The **1:1** (or one – to – one) relationship should be rare and can be optional in any relational database design.
* The **M:N** (or many – to – many) relationship is not directly supported in a relational environment, but it can be changed into a 1:M relationship.

The **Entity Relationship Diagram (ERD)** represents the conceptual database as viewed by the end – user using graphical representations to model database components such as the entity, its attributes, and even their relationships.

Figure 10 shows two representations of the ERD using the Chen notation and the Crow’s Foot notation.



**Figure 10.** Representations of the ERD

**Chen notation** was based on Peter Chen’s landmark paper. In this notation, the connectivities are written next to each entity box. Relationships are represented by a diamond connected to the related entities through a relationship line. The relationship name is written inside the diamond.

On the other hand, **Crow’s Foot notation** was derived from the three – pronged symbol used to represent the “many” side of the relationship. Further examination on the basic Crow’s Foot

ERD in Figure 10 shows that the connectivities are represented by symbols. For example, the “1” is represented by a short line segment, and the “M” is represented by the three-pronged “crow’s foot.” In this example, the relationship name is written above the relationship line.

* **HOW TO DEVELOP AN ERD?**

Creating a good Entity Relationship Diagram would always depend on how a database designer understood the business rules that needs to be implemented within the database system to be developed.

Here are some simple steps how to develop an ERD:

***Step 1:*** Identify the entities.

***Step 2:*** Identify the attributes of each entities. The primary key of each entity must be identified as well.

***Step 3:*** Determine the relationship that exists among entities. This also means that the connectivity and cardinality should be also determined here.

***Step 4:*** Draw the actual ERD using either Chen Notation or the Crow’s Foot Notation.

**Example:** Create an ERD for the business rules:

* Each BOOK is published by one and only one PUBLISHER.
* Each PUBLISHER can publish many BOOKS.

**Solution:** Based from the given business rules, let us try to develop an ERD based from the four basic steps:

***Step 1:*** From the given business rules, it is obvious that only two entities can be identified; the **BOOK** and **PUBLISHER**.

***Step 2:*** For each entity, the following attributes may be identified:

**BOOK** **PUBLISHER**

*Book\_ID (PK)*  *Publisher\_ID (PK)*

Title Publisher\_Name

Author Publisher\_Address

Genre

Publisher\_Name

Date\_Published

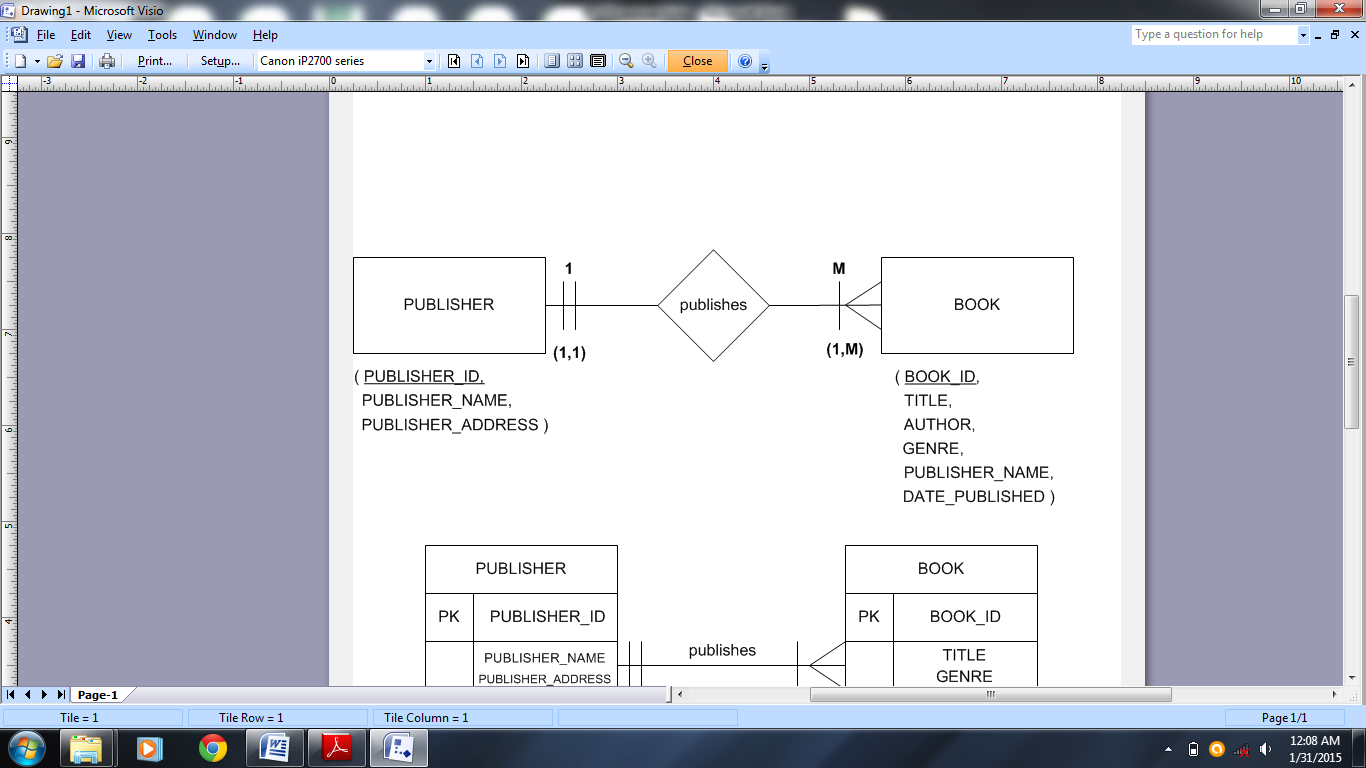
***Step 3:*** Based from the given business rules, it is obvious that a 1:M (one – to – many) relationship can be established. This now serves as the connectivity of the ERD. **Connectivity** is used to describe the relationship classification.

But what about the ERD’s cardinality? The term **cardinality** expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (***x,y***). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities.

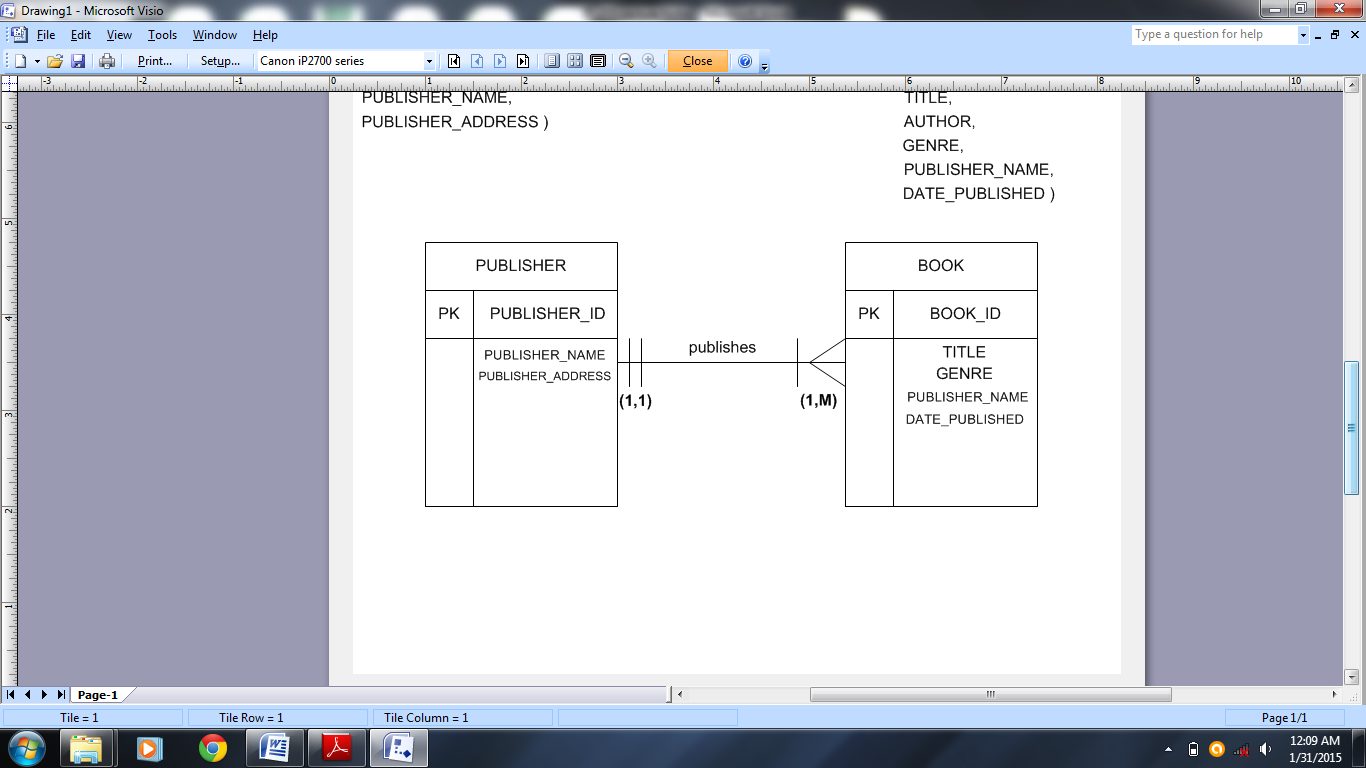
With that being said, the following cardinalities may be achieved; **(1,N)** for **BOOK** and **(1,1)** for **PUBLISHER**. For both entities, they exhibit a mandatory participation. This means that means that one entity occurrence **requires**a corresponding entity occurrence in a particular relationship.

***Step 4:*** Finally, based from the previous steps, the following ERD can be developed:

**Using Chen Notation:**



**Using Crow’s Foot Nation:**



\*\*\***END OF CHAPTER 3**\*\*\*

**Chapter 4**

**INTRODUCTION TO SQL**

**Objectives:**

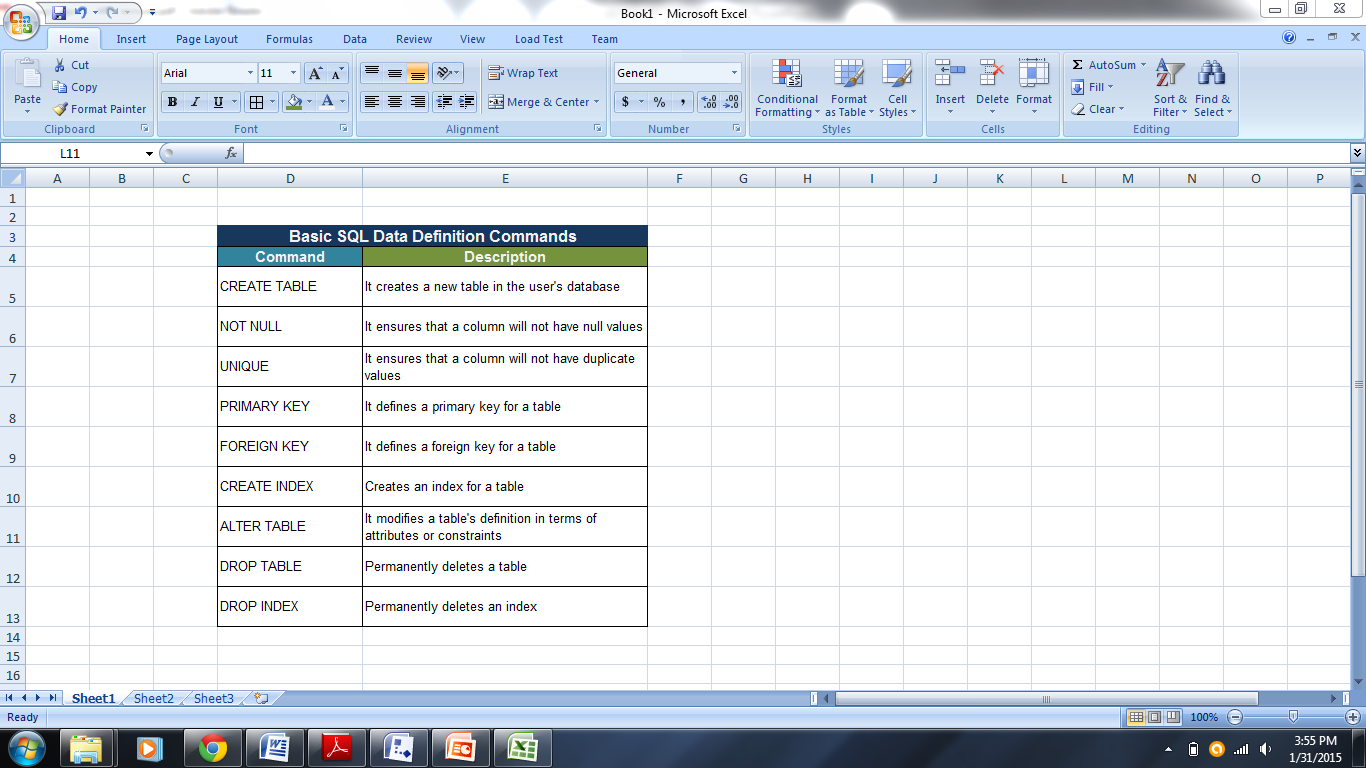
At the end of this chapter, the students should be able to:

* Learn the basic commands and functions of SQL.
* Use SQL in data definition
* Use SQL in data manipulation (add, update, delete and retrieve data).
* **WHAT IS SQL?**

Structure Query Language (SQL) is composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration, and query the database to extract useful information.

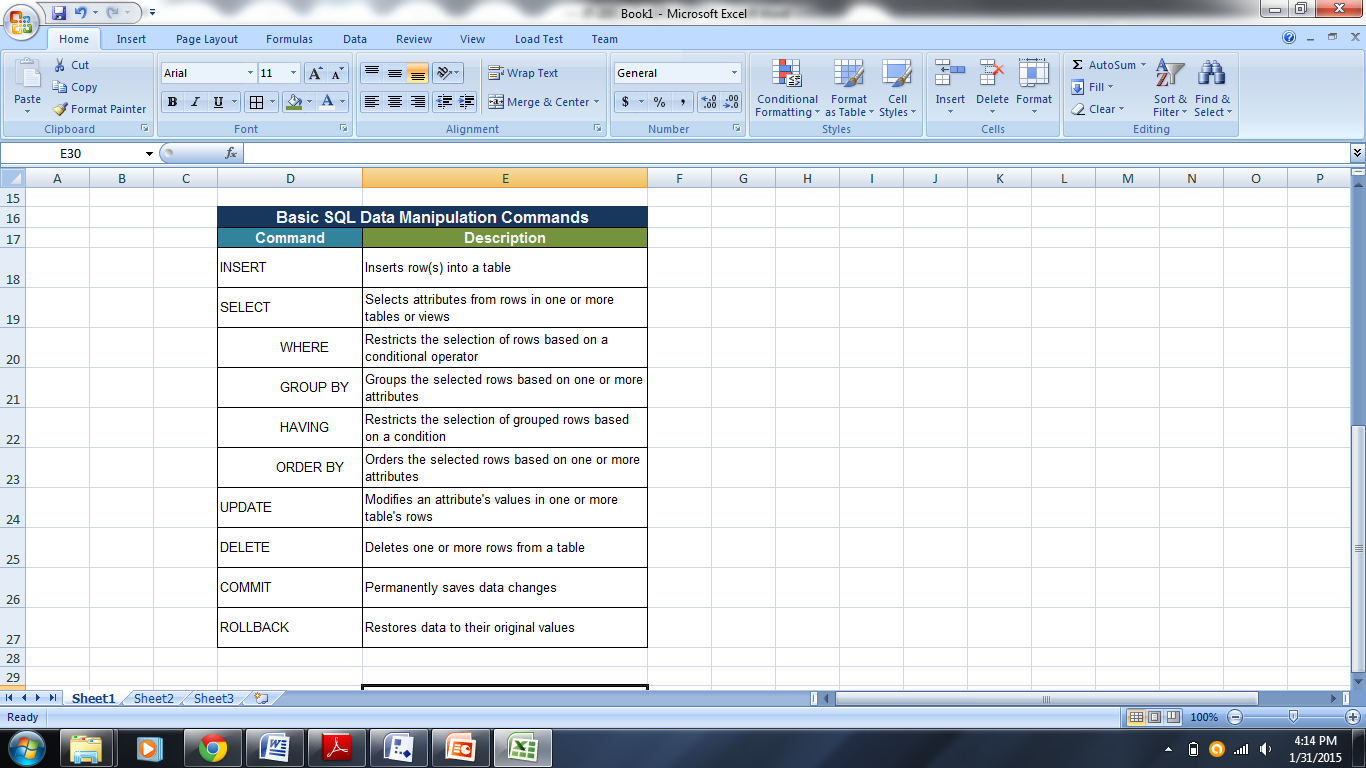
SQL functions fit into two broad categories:

* *Data Definition Language (DDL).* It is used to build and modify structure of your tables and other objects in the database.



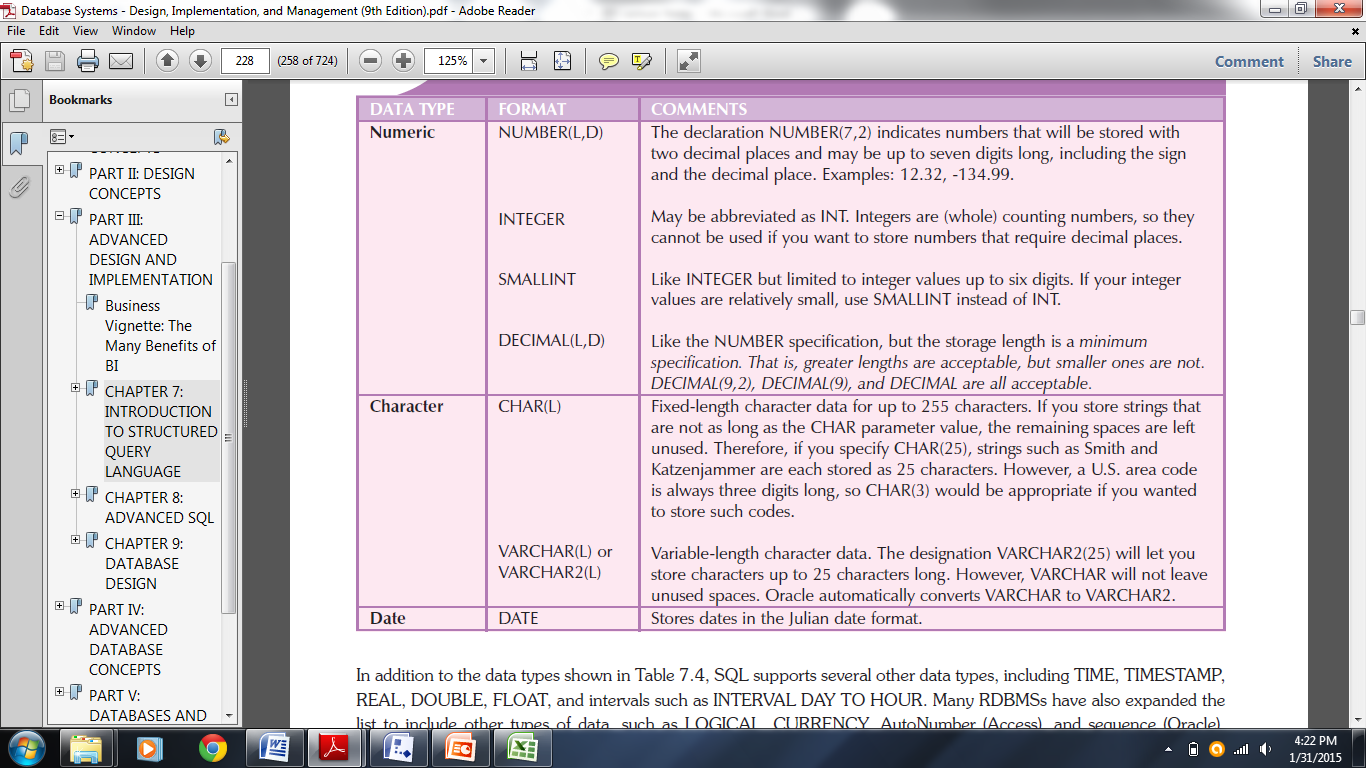
**Table 3.** Basic SQL Data Definition Commands

* *Data Manipulation Language (DML).* It is used to work with data in tables.



**Table 3.** Basic SQL Data Manipulation Commands

SQL is not a programming language, but since its standard where prescribe by the American National Standards Institute (ANSI), it also has a prescribed syntax and data types as shown in Table 4.



**Table 4.** Common SQL Data Types

Standards for SQL exist. However, the SQL that can be used on each one of the major RDBMS today is in different flavors. This is due to two reasons: 1) the SQL command standard is fairly complex, and it is not practical to implement the entire standard, and 2) each database vendor needs a way to differentiate its product from others.

Some common relational database management systems that use SQL are: Oracle, Microsoft SQL Server, Microsoft Access, MySQL, and others. The succeeding sections will tackle the basics of SQL programming focuses on DDL and DML.

* **SQL PROGRAMMING BASICS**

The use of Structured Query Language (SQL) allows a programmer to access or manipulate records in the database. And with this being said, the following reminders in SQL programming are very important:

* An SQL statement always ends with a semi – colon. No matter how many lines were actually used for the entire SQL statement, having a semi – colon at the end will make it as a single SQL statement.
* SQL is not case – sensitive. Even camel – casing an SQL statement is fine, as long as the correct SQL statement is encoded.

To better understand SQL programming, let us try to explain the proper syntax in accordance to the CRUD (Create, Read, Update, Delete) concept in database.

* **SQL CREATE COMMANDS**

The following are some of the basic CREATE commands used in SQL:

* ***CREATE DATABASE***

– It is the SQL statement used to create a database within the SQL.

*Syntax:* **CREATE DATABASE** (database name);

*Example:* **CREATE DATABASE** dbms1;

Whenever the database has been created, use the USE DATABASE command to access it. From this, creating a table within that database is now possible.

*Example:* **USE DATABASE** dbms1;

* ***CREATE TABLE***

– It is the SQL statement used to create a table within the database selected.

*Syntax:* **CREATE TABLE** (table name)

(

column\_name1 data\_type(size or length),

column\_name2 data type(size or length),

. . . .

);

*Example:* **CREATE TABLE** student

(

sno int auto\_increment,

fname varchar(50) not null,

lname varchar(50) not null,

course varchar(10),

year int(3),

primary key(sno)

);

Notice that words such as **auto\_increment** and **not null** appears in the whole SQL statement. **Auto increment** tells that the defined column name will be having an automatic value coming from the DBMS itself since its data type is an integer. And since no size or length that was defined for the **sno** column name, it will adopt the maximum size for integer which is 11 for MySQL.

**Not null** in the SQL statement tells that the column defined should have no null (blank) values for they are considered as required fields in the table when inserting a record.

And notice that in the SQL statement, the field **sno** was identified as the table’s primary key. As defined in the previous chapter, a primary key serves as the unique identifier in every record within a table.

* **INSERTING RECORDS USING SQL**

Now that a table has been create in a specific database, that following are some ways to insert a record into that table:

*Syntax #1 (for single record):*

**INSERT** **INTO** table\_name (columns needs to be inserted)

**VALUES** (values that will be inserted into each column);

*Example #1:*

**INSERT INTO** student (fname,lname,course,year)

**VALUES** (‘Christian’,’Fajardo’,’MACE’,1);

*Syntax #2 (for multiple records):*

**INSERT** **INTO** table\_name (columns needs to be inserted)

**VALUES** (values will be inserted as record 1),

(values will be inserted as record 2),

. . . .;

*Example #2:*

**INSERT INTO** student (fname,lname,course,year)

**VALUES** (‘Baden Darwin’,’Carranza’,’MIT’,2),

(‘Ma Sheryl’,’Rebualos’,’MSSME’,1),

(‘Carla Carmela’,’Perez’,’MIT’,1);

For both approaches in inserting a record, noticed that the sno column was not included in the columns that needs to be inserted a value. That is because the sno column was actually auto incremented. This means if the following records were inserted the correct order, Christian Fajardo should be sno # 1, while Ma Sheryl Rebualos should be sno #3.



**Figure 11.** Records in the “student” table

* **COMMON SQL READ COMMANDS**

The term “Read” in database means to view something either in the database or at the tables itself. It could be either the table’s description in accordance with its data dictionary or with the table’s records.

The following SQL statements are commonly used:

* *The* ***DESCRIBE*** *Statement*
* It provides information about the columns of the table such as its data type and size.

*Syntax:* **DESCRIBE** (table\_name);

*Example:* **DESCRIBE** dbms1;

* ***SHOW DATABASES***
* It shows the list of databases in the SQL server host.

*Syntax:* **SHOW DATABASES;**

* ***SHOW TABLES***
* It shows the lists of non – temporary tables in a given database. The term “non – temporary table” means that they can delete within the database without affecting the whole SQL server host. Just make sure that before using this statement, the database where the table is located is also in use using the **USE DATABASE** statement.

*Syntax:* **SHOW TABLES;**

The next section will explain the SELECT statement, which is one of the most common SQL command specifically for queries.

* **THE “SELECT” STATEMENT**

The SELECT statement is commonly used in SQL to retrieve rows selected from one or more tables. This SQL statement is can be used in many ways to request a query from the table. And following are some of the most common examples on how to use the SELECT statement:

*Syntax #1:*

**SELECT \* FROM** table\_name;

*Syntax #2:*

**SELECT** (specific column\_names that is separated by a comma)

**FROM** table\_name;

*Syntax #3:*

**SELECT** (specific column\_names that is separated by a comma)

**FROM** table\_name

**WHERE** conditions, if any;

*Syntax #4:*

**SELECT** (specific column\_names that is separated by a comma)

**FROM** table\_name

**WHERE** conditions, if any

**ORDER BY** column\_name [**ASC**/**DESC**];

Notice that on each SELECT statement, the FROM clause is always included. Other clauses such as WHERE and ORDER BY can be added for a more specific table query using SQL. The asterisk (**\***) in a SQL statement tells that all columns are requested to be shown back as a result of a table query.

Example: Given the following table below, do the following table queries as specified:

*Table Name: Student*



1. Show all records
2. Show all records having the column names “Last Name”, ”First Name”, and “Course” only.
3. Show all records displaying the “First Name” and “Last Name” where their course is “MIT".
4. Show all records where their year is not “2” in a descending order.

Solution:

1. Since all records are required to be displayed from the “Student” table, then the following SQL statement should be entered:

**SELECT \* FROM** Student;

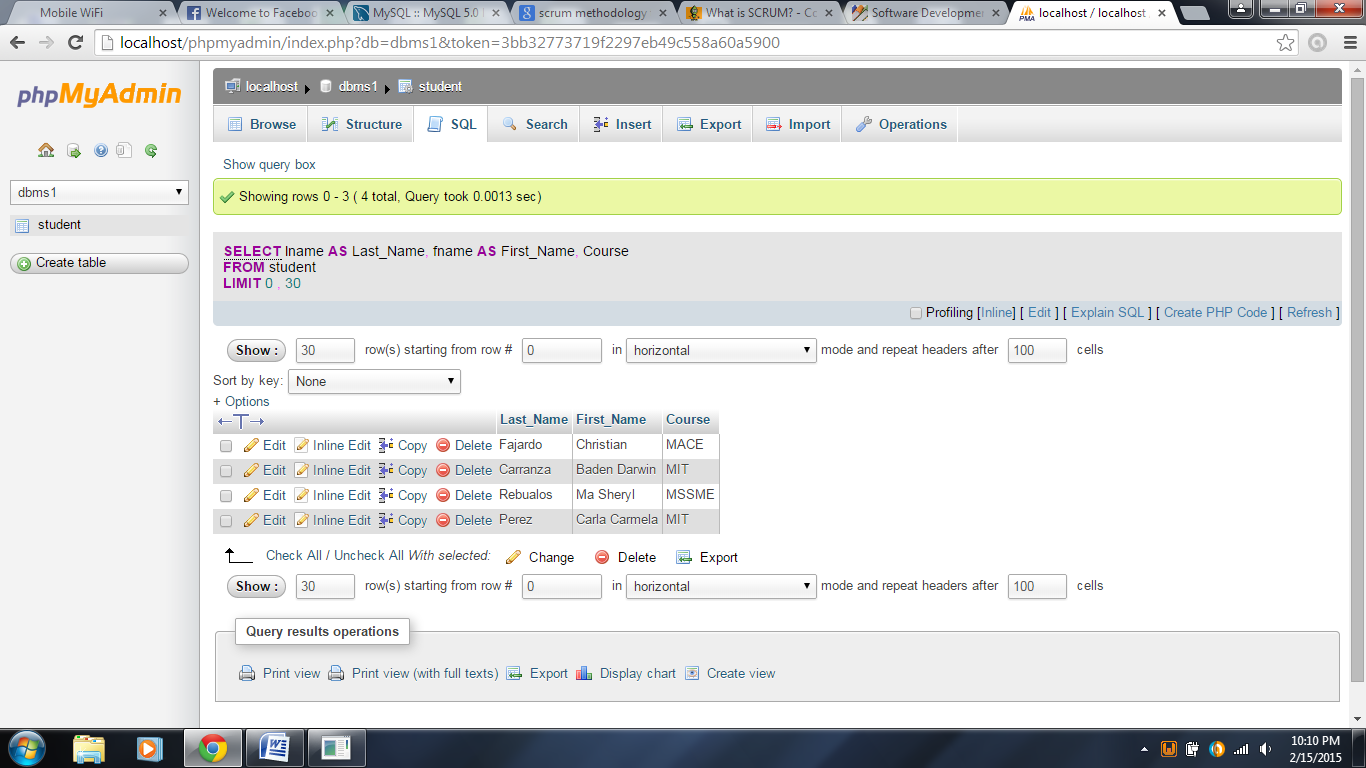
If done correctly, then the query result set should return the same table as the given table above.

1. From 5 columns, the query result set should only return the columns “lname”, “fname”, and “course” for all records from the “Student” table. And since column names must be renamed as “Last Name”, ”First Name” , and “Course” respectively, then following SQL statement should be entered:

**SELECT** lname **AS** Last\_Name,fname **AS** First\_Name,Course

**FROM** Student;

If done correctly, the following table result should return as displayed below:



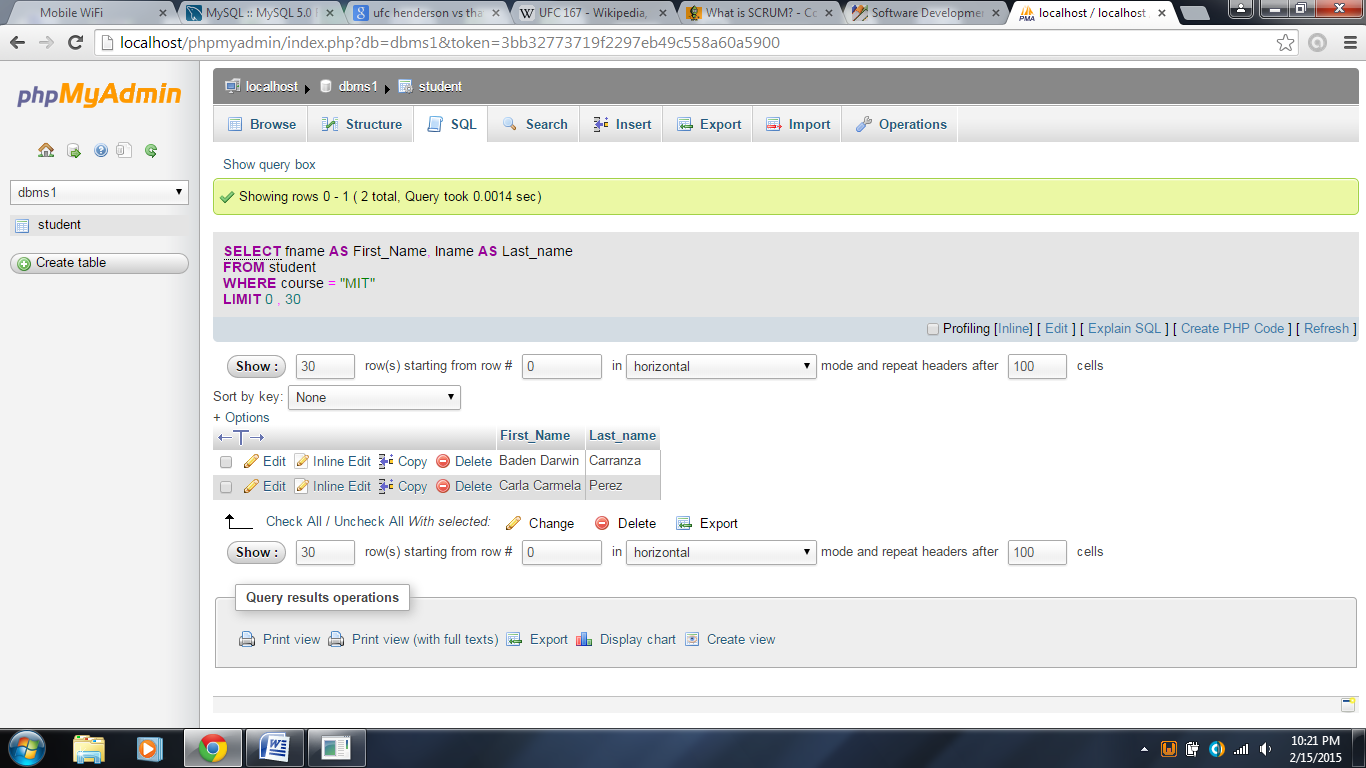
1. The query result set should return records showing the “fname” as “First\_Name” and “lname” as “Last\_Name” where their course is “MIT”. Therefore, the following SQL statement should be entered:

**SELECT** fname **AS** First\_Name,lname **AS** Last\_Name

**FROM** Student

**WHERE** course=‘MIT’;

If done correctly, then the following table result should appear as displayed below:



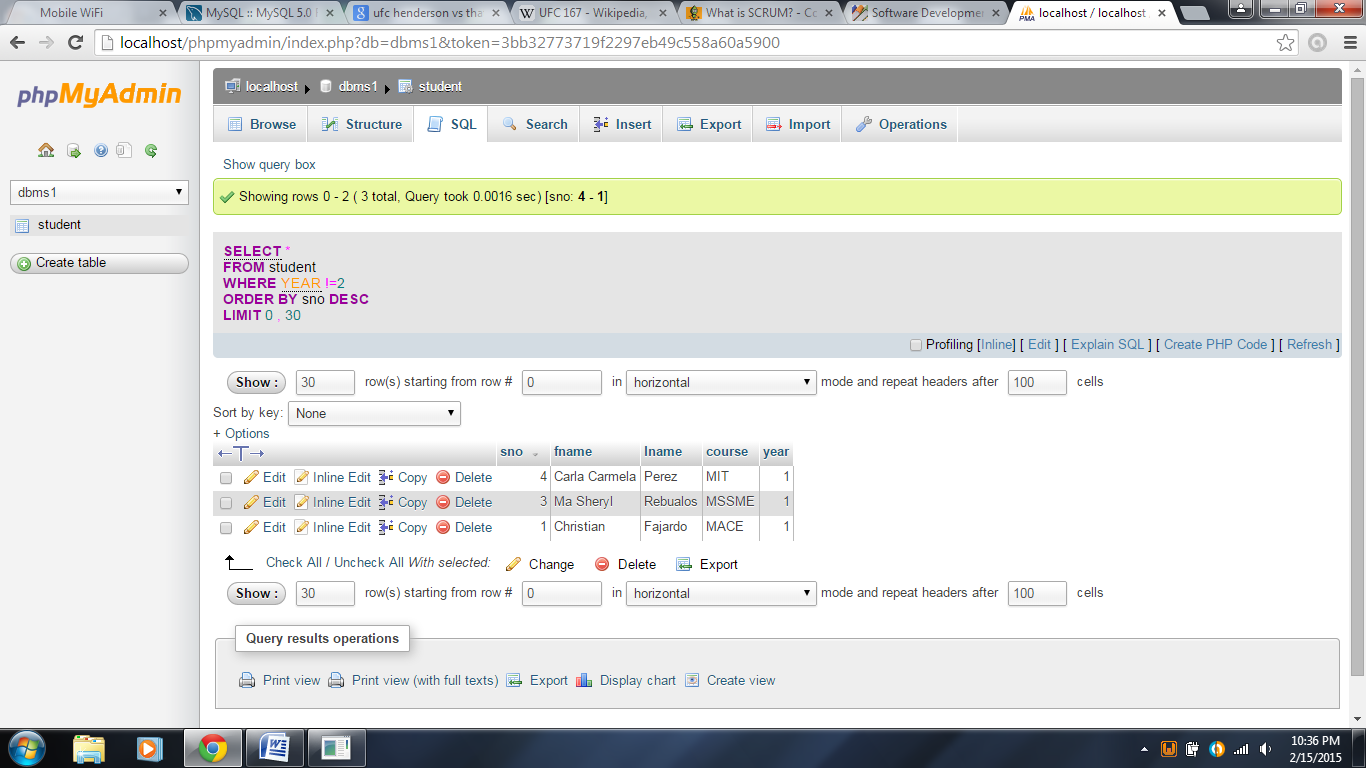
1. The query result set should return all records with all columns where their year is not “2”. Therefore, the following SQL statement should be entered:

**SELECT \* FROM** Student

**WHERE** year!=2

**ORDER** **BY** sno **DESC**;

If done correctly, the table result below should appear:



* **USING SQL WILDCARDS**

In SQL, wildcards are used for searching records within the table from a specific domain. A wildcard character can be used to substitute for any character in a string.

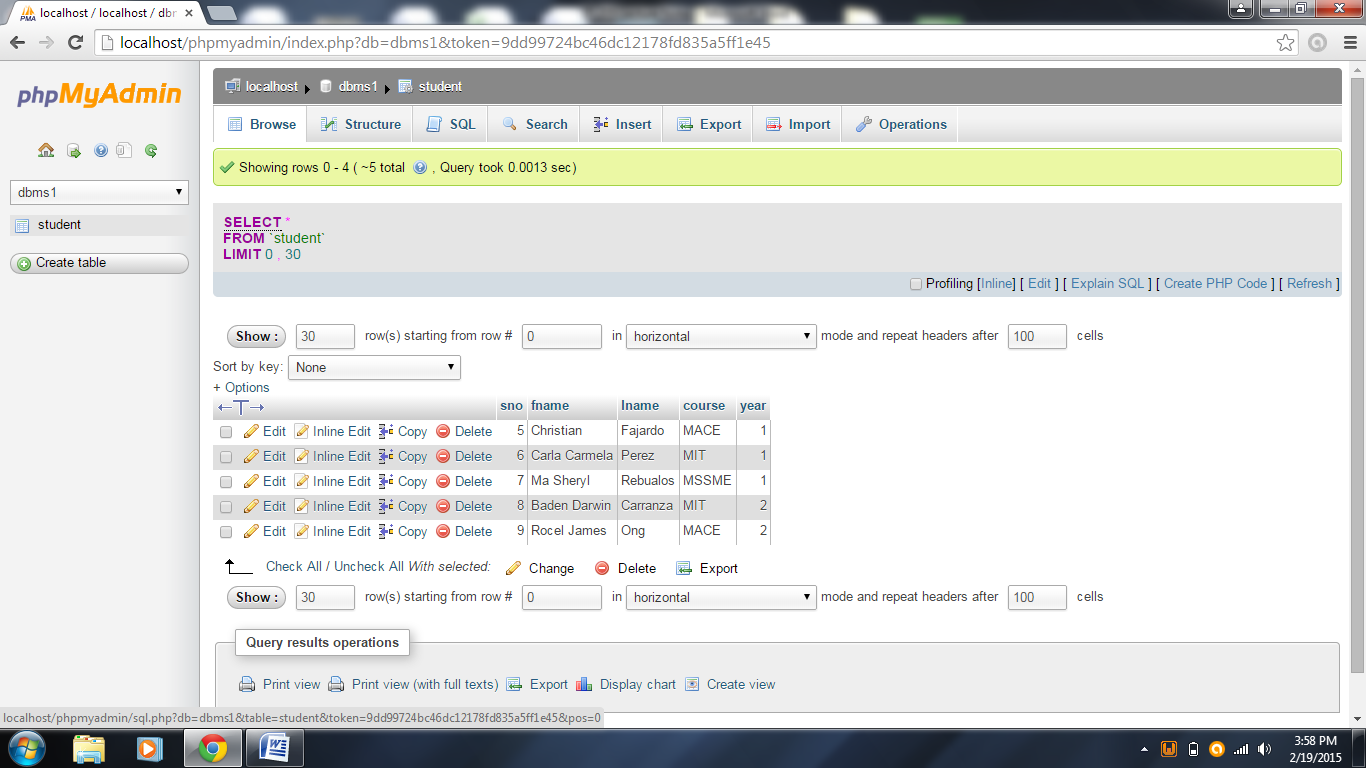
The table below shows wildcard characters are used in SQL:

|  |  |
| --- | --- |
| **Wildcard Character** | **Description** |
| % | A substitute for zero or more characters |
| \_ | A substitute for exactly one character |
| [character list] | Any single character in the list |
| [^character list]  or  [!character list] | Any single character NOT in the list |

Whichever SQL wildcard is being used, there should be a LIKE operator at each statement.

Example: Given the following table below, show all records wherein their first names end with letter “n”. Sort the results by their year if possible.

*Table name: Student*

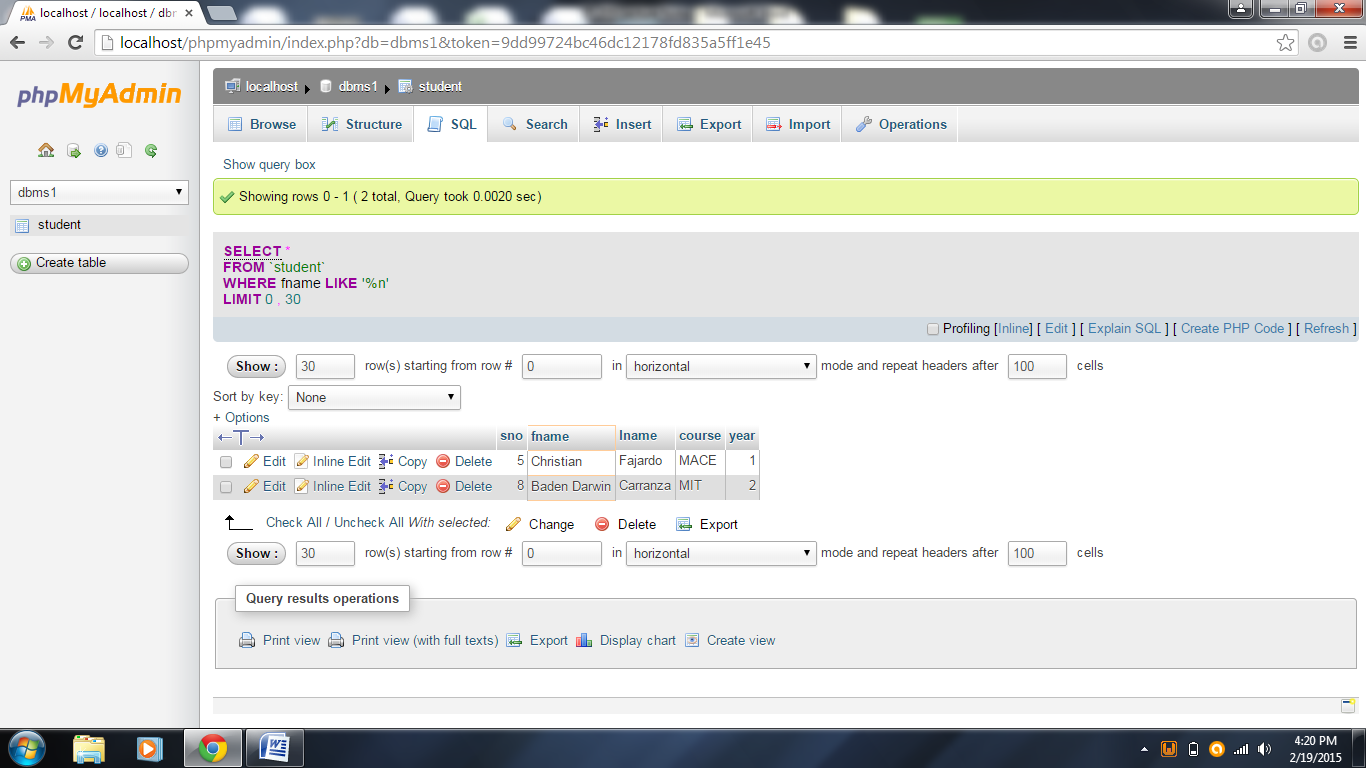


Solution: Since the required record should end at the letter “n” on its first name, the following SQL statement could be used:

**SELECT** \* **FROM** Student

**WHERE** fname **LIKE** ‘%n’;

When done correctly, the following table result should appear:



* **COMMON SQL MODIFY COMMANDS**

It is usual for a database system that some data entries can be modified for whatever reason it could be. Nevertheless, here some common modify commands used in SQL:

* THE ***“UPDATE”*** STATEMENT

– It is used to update existing records in the table.

*Syntax:* **UPDATE** table name

**SET** column\_name1=new\_value

**WHERE** reference\_column\_name=corresponding\_value;

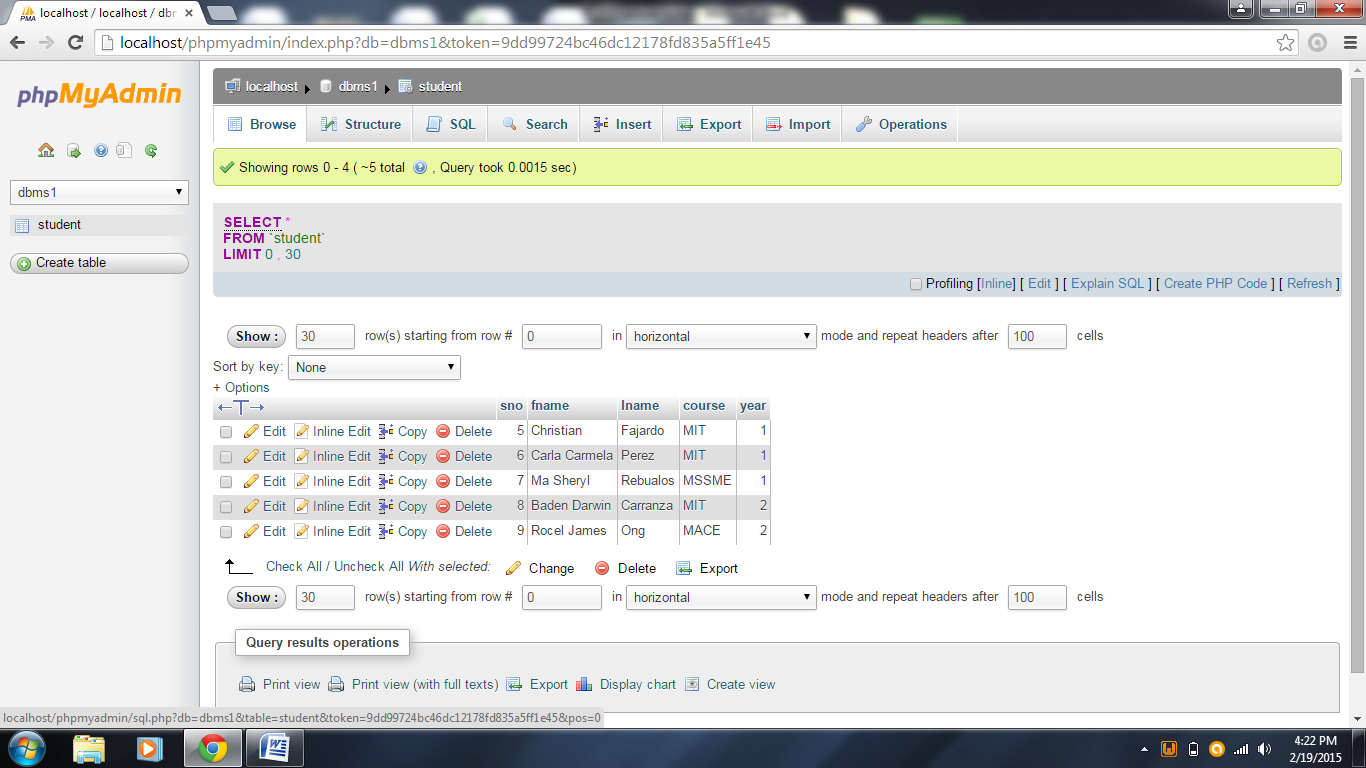
It is very important that a WHERE clause is always included whenever using the UPDATE statement. Otherwise, all records will be updated with the new value being entered within the UPDATE statement.

*Example:* **UPDATE** Student

**SET** course=’MIT’

**WHERE** sno=5;

When correctly done, the course should be changed from ‘MACE’ to ‘MIT’ as shown from the table result below.



* THE ***“ALTER TABLE”*** STATEMENT

– This statement allows changing the structure of an existing table such as adding a new column, renaming an existing column, or even modify indexes within the table that needs to be altered.

*Syntax #1 (Adding a new column):*

**ALTER TABLE** table\_name

**ADD** column\_name data\_type(size or length);

*Syntax #2 (Modifying an existing column):*

**ALTER TABLE** table\_name

**ALTER COLUMN** \_new\_column\_name data\_type(size or length);

*Syntax #3 (Deleting an existing column):*

**ALTER TABLE** table\_name

**DROP COLUMN** column\_name;

* **COMMON SQL DELETE COMMANDS**

The following SQL statements or commands are normally used to delete a record, a table, or even a database:

* ***DROP TABLE***

– This command removes one or more tables. Be careful when using this statement though, because all table data and table definition will be removed as well.

*Syntax:*

**DROP TABLE** table\_name;

* ***DROP DATABASE***

– This command removes all tables in the database and the database itself.

*Syntax:*

**DROP DATABASE** database\_name;

* THE ***“DELETE”*** STATEMENT

– This command removes a record within the table.

*Syntax:*

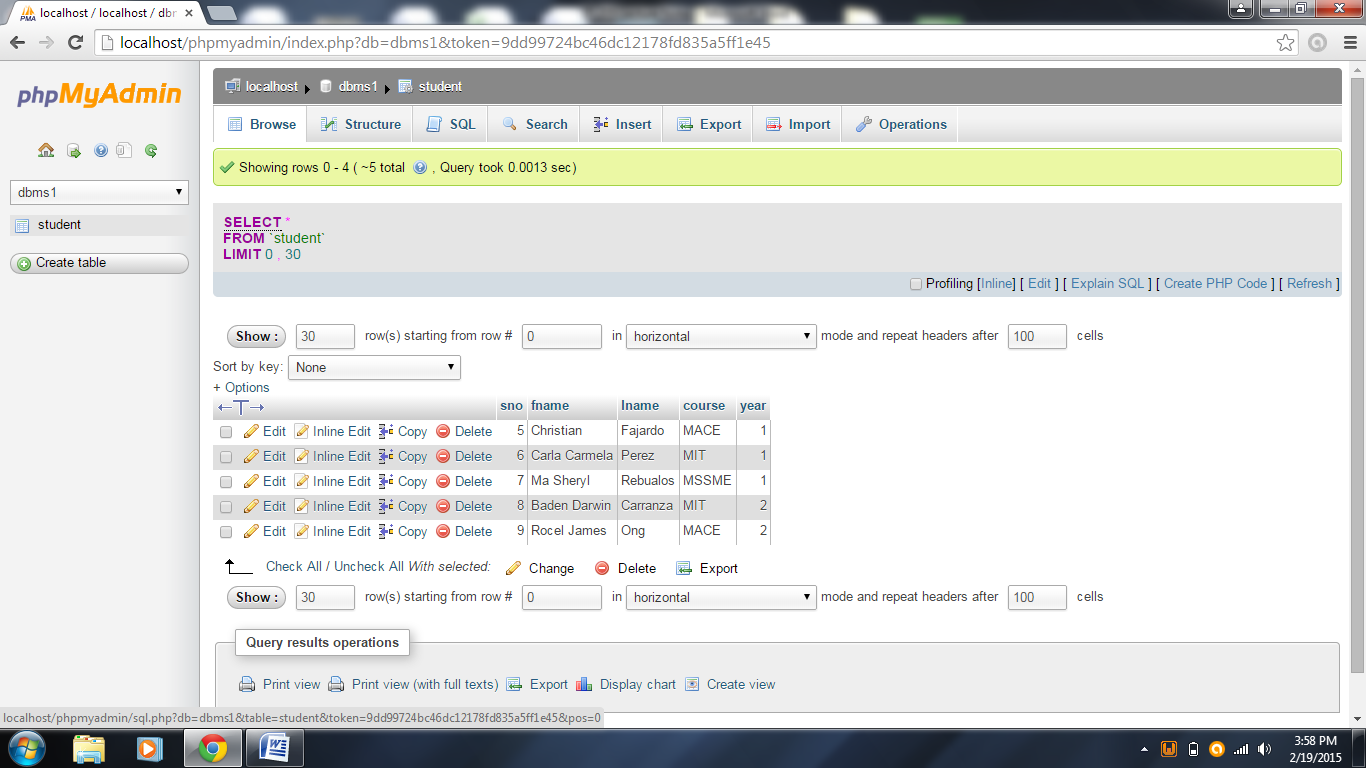
**DELETE FROM** table\_name

**WHERE** column\_referencing \_condition;

Make sure that whenever invoking this statement, the WHERE clause should be always included, otherwise, all records within the table will be removed.

Example: Given the following table below, remove the record of “Baden Darwin Carranza”.

*Table name: Student*

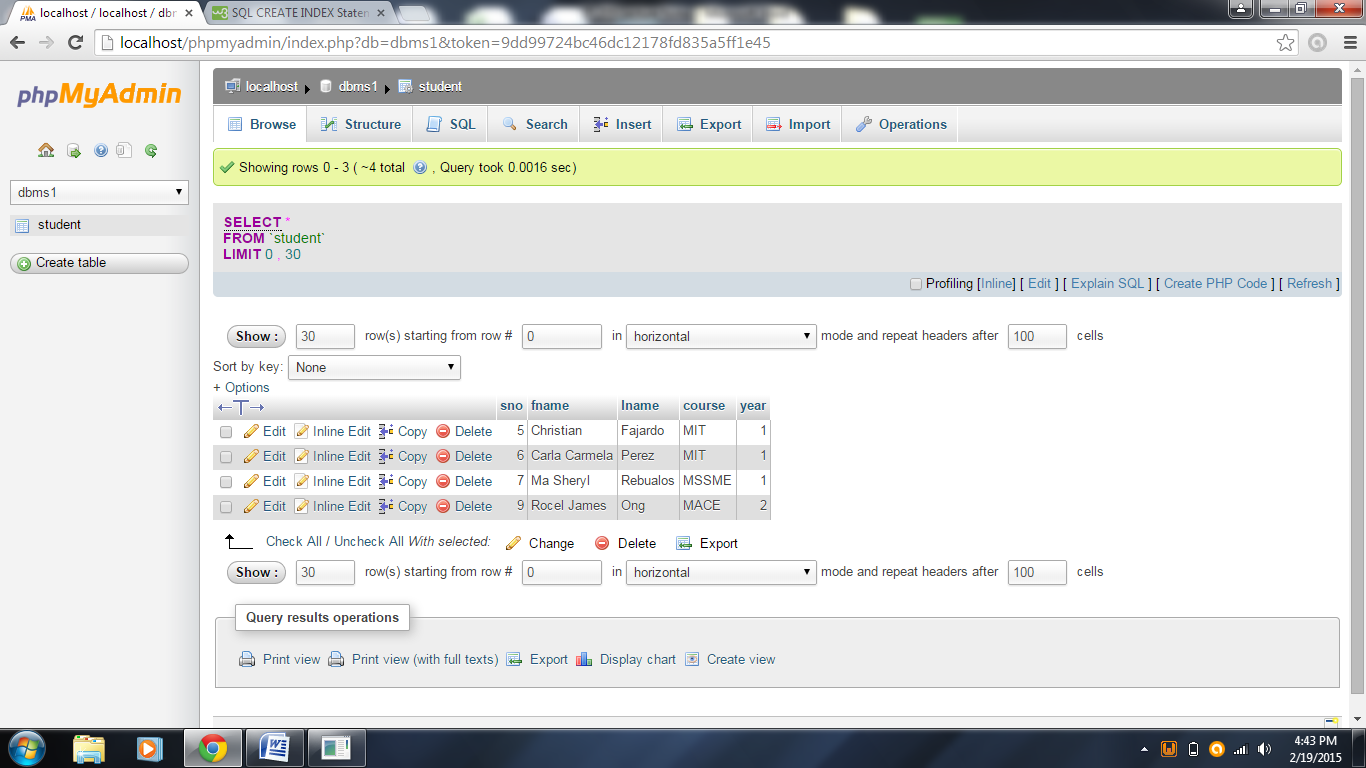


Solution: Assuming that the sno is the table’s primary key, which can be used as the column reference to delete “Baden Darwin Carranza’s” record only. With that being said, the following SQL statement should be entered:

**DELETE FROM** Student

**WHERE** sno=8;

If done correctly, the record will be deleted and the following table result should appear once all records within the table is viewed.



\*\*\***END OF CHAPTER 4**\*\*\*