CS708 Lecture Notes

Visual Basic.NET Programming

Object-Oriented Programming
Database Basics

(Part I)
(Lecture Notes 4A)

Prof. Abel Angel Rodriguez
CHAPTER 1. REVIEW OF DATABASE PRINCIPLES ................................................................. 4

1.1 Relational Database Review ..................................................................................................... 4
   1.1.1 Introduction ............................................................................................................................. 4
   1.1.2 Database Management System (DBMS) ................................................................................. 4
   1.1.3 Database Application .............................................................................................................. 5
   1.1.4 Introduction to the Database Development Process ................................................................. 6
   1.1.5 The People Involved in Database Development Process ...................................................... 7

1.2 Database and Client/Server Architecture .................................................................................. 8
   1.2.1 Client/Server Architecture ...................................................................................................... 8
   1.2.2 Client/Server Operating Systems ......................................................................................... 9
   1.2.3 Client/Server Architecture & Application Development ......................................................... 10
   1.2.4 Desktop Database (Single-Tier Client/Server) ..................................................................... 11
   1.2.5 Database Servers (Two-Tier Client/Server) ........................................................................... 13
   1.2.6 Web-Based Client/Server, the Internet and Database Applications ....................................... 17
   1.2.7 Intranet & Web Development for Internal Database Applications ...................................... 17

CHAPTER 2. COMPONENTS OR A RELATIONAL DBMS ............................................................ 18

2.1 Components of a Relational DBMS ........................................................................................ 18
   2.1.1 The Components that make up a Database Management System ......................................... 18

2.2 Relational Tables ..................................................................................................................... 18
   2.2.1 Table Basics .......................................................................................................................... 18
   2.2.2 Table Relationship .............................................................................................................. 20

2.3 Introduction to Queries, Server-Side Processing, View and Reports ......................................... 22
   2.3.1 Introduction ............................................................................................................................ 22
   2.3.2 Advanced Server-Side Processing SQL Functionality ............................................................ 22
   2.3.3 Views ...................................................................................................................................... 23
   2.3.4 Reports ................................................................................................................................. 23

CHAPTER 3 BUILDING SQL QUERIES .................................................................................. 24

3.1 Building SQL Statements (Introduction) .................................................................................. 24
   3.1.1 Introduction ............................................................................................................................ 24

3.2 SELECT Statement & Associated Clauses ............................................................................... 26
   3.2.1 SELECT Statement ................................................................................................................ 26
   3.2.2 WHERE Clause ...................................................................................................................... 28
   3.2.3 The ORDER BY Clause ........................................................................................................ 31
   3.2.4 The GROUP BY Clause ....................................................................................................... 32
   3.2.5 SQL Aggregated Functions ................................................................................................. 32

3.3 UPDATE Statement & Associated Clauses ................................................................................ 34
   3.3.1 UPDATE Statement .............................................................................................................. 34

3.4 INSERT Statement & Associated Clauses ................................................................................ 35
   3.4.1 INSERT Statement ................................................................................................................. 35

3.5 DELETE Statement & Associated Clauses ................................................................................ 36
   3.5.1 DELETE Statement .............................................................................................................. 36
Chapter I. Review of Database Principles

1.1 Relational Database Review

1.1.1 Introduction

- A database is an organized collection of related data, which can be manipulated, sorted, analyzed and displayed quickly and efficiently.
- The term relational means that the data is stored in a table consisting of columns (called fields) and rows (called records).
- A database is nothing more than a computerized record-keeping system.
- The users of a database are given the tools to perform a variety of operations on the storage, such as:
  - Inserting, retrieving and updating records into the database.
  - Deleting a record from the database.
  - Removing or deleting the database tables or files.
- Examples of current databases are:
  - Business databases which contain information on employees, payrolls, customers, products, inventories, and sales.
  - Databases in schools contain information on courses, students, teachers, and grades.
  - A video store which need to maintain records of video rental/return transactions and customer records.
  - Banks are big users of databases; they are used to store customer information, & all transaction records etc.
  - The telephone companies store telephone account records, transactions & invoices etc.
  - In the new age of the Internet, Web Servers use databases to store e-commerce transactions, products, search engines directory information, etc. This has lead to new storage technology such as Data Warehousing etc.

1.1.2 Database Management System (DBMS)

- The software Program used to design, create databases are known as a Database Management Systems. Examples of such Programs are:

  Database Servers:
  - Oracle
  - Microsoft SQL Server Enterprise
  - Sybase SQL Server
  - IBM DB2

  Desktop Database:
  - (New) MSDE – Microsoft SQL Desktop Engine (Desktop version of MS SQL Server, limited # users and storage)
  - Microsoft Access (a desktop DBMS)
  - FileMaker Pro

- A DBMS is primarily composed of the following components:
  - Tables
  - Queries (SQL)
  - Reports

- A DBMS usually provides all the required tools to create a complete database that can include:
  - The storage facility or database engine itself.
  - A programming language (SQL) to retrieve, inserts, & update the data stored in the database engine.
  - Custom (native-code) programming language that gives you the power to write more complex Server-side processing.
  - Combination of SQL & Custom native code, for example:
    1. In Microsoft SQL Server with have the following:
      - Stored Procedures – a set of one or more SQL statement grouped together. Can include custom language code
      - Triggers – a special type of Stored Procedure that executes automatically.
2. In Microsoft Access we have the following:
   - *Visual Basic for Applications* (VBA) Visual Basic (without the Forms) embedded within Access.
   - Application Programming Interface (API) that allows you to interface native-code DBMS written functions with to the OS & other systems.

   The diagram below illustrates a DBMS and a client application that is requesting data from the database engine or DBMS:

   ![Diagram of DBMS and Client Application](image)

   Note that the client application is NOT part of the DBMS. We will discuss this next.

### 1.1.3 Database Application

A Database Application is a program (or a set of related programs) that on behalf of a user will perform operations on DBMS, such as create, retrieve, update or delete data. Examples of such Programs are:

- ATM machines
- Business Applications
- Program used by a telephone operator to retrieve information
- Program used by Bank Tellers to perform transactions on behalf of the customers.
- Program used in Video stores to manage video rental & return transactions and invoices.
- Programs used in a supermarket to process the sales of food items.
- Department store programs etc.

The list of these programs goes on and on. Database applications are the most common type of programs. Makes sense. We use computers to process and manage data. Data is stored in a Database. **GET IT?**

- A Database Application is composed of two parts 1) the Front-End or Client Application and 2) the DBMS engine itself.
- Diagram bellows illustrates the components of a Database Application:
1.1.4 Introduction to the Database Development Process

- Now that we know that there are applications known as *Database Applications* that interact with databases on behalf of a user.
- But how are these applications created?
- Well a Database Application involves two parts, the Front-End Client Application and the DBMS Application.
- Database applications can be written using the following methods:

1. **Method I** – Write the front-end GUI and processing logic using and external programming language to connect to the DBMS. The DBMS only handle the database processing part.
   - With today's business demands and so many diverse business systems, and powerful programming tools available, most DBMS are primarily used to manage the Database only, while the front-end (GUI) and complex processing is done using other programming tools such as Visual Basics.NET, C#, Java, & C++. Visual Basic is a popular tool used to create applications which require database connectivity. Writing the front end in Visual Basics is fast and give you more flexibility and power.

2. **Method II** – Use Client/Server Internet technology or Web Development to create the database application.
   - The Internet or World Wide Web has created a new way of writing database applications. In Web development, the front-end GUI or Web Browser, and the application is written with tools such as HTML, VBScript, Java Applets, JavaScript etc. and the database back-end using a Web Server and DBMS system. This is now becoming the standard.
Steps required to Create a Database Application

- In this lecture we will go through an overview of the overall steps required for the database development process. These steps are:
  
  - **Analysis** – Analyze the business or organization in order to determine requirements for developing database
  
  - **Design** – Organize all the requirements and produce a detailed specification of all data, forms, reports, displays, and processing or business rules. This includes purchase of all hardware, software etc.
  
  - **Implementation** – Create the database, tables, view, stored procedures etc. Create Front-End Client programs. Test and install system, train users and document the installation and Standard Operation Procedures (SOP)
  
  - **Maintenance\Administration** – Monitor, backup & repair the system. Periodically audit system to ensure it continues to meet requirements. Have a disaster/recovery system in place.

1.1.5 The People Involved in Database Development Process

The Project

- Before listing the people involved we need to first explain that the entire development process is implemented via a project.
- **Project** - Planned list of tasks from beginning to end to reach and objective
- A project requires a **Project Manager** or leader is the person responsible for planning and monitoring the project from beginning to end.
- A good project manager will possess skills in leadership, management, customer relations and communications, technical skills, conflict management, team building, risk and change management.

The Team

- The team involves the two groups: 1) Front-End Client Program developers, 2) Back-End Database Developers.

Database Development Team

- The Database Development team can included the following personnel:
  
  - **Systems Analysts** – Analyze the business structure and situation and identify the needs for the database or information system to meet the problems or opportunity of the business. This is the person who interfaces with the Business Group of the business and compiles the requirements and business rules. Create Data Modeling diagrams etc.
  
  - **Database Analysts** – Based on the information provide by the Systems Analysts, these individual determine the requirements and design for the database itself. Create Entity Diagrams etc.
  
  - **Database Developers or Programmers** – Design and write the **Back-End Database Application**: Tables, Stored Procedures, and View etc.
  
  - **Database Administrators** – Responsible for the day to day operations of the database. They create databases within the DBMS, ensure that database is backed up and the proper disaster/recovery measures are in place. Ensure database is secure, and ensure consistency and integrity across all databases.
  
  - **Other Technical Experts** – These are other IT professionals in the organization which play a supporting role, such as Network Administrators (LAN/WAN), Server Administrators, Security Experts etc.

Front-End Client Development Team

- The Client Development team can included the following personnel:
  
  - **Business Object Developer/Designer** – Design and program the objects needed for the application.
  
  - **User-Interface Developers** – Design and write the Front-End GUI, that will interact with the user.
  
  - **Other Technical Experts** – These are other IT professionals in the organization which play a supporting role, such as Network Administrators (LAN/WAN), Server Administrators, Security Experts etc.
1.2 Database and Client/Server Architecture

1.2.1 Client/Server Architecture

Introduction

- Client/Server Technology is the standard that has been used when creating network applications and systems.
- The Client/Server definition is as follows:
  - **Client** – Process or executable program that makes requests to another process (executable)
  - **Server** – Process or executable program that services or complies with the request made by a client process, thus the name server.

- The diagram bellow illustrates the Client/Server definitions. Note how the client process makes a request to the Server Process. The server process handle the request for the client:

![Client Process Memory Space](image)

Client/Server Interaction

- Note that the name Client and Server have been adopted for many other systems in the field of computer science. For example, the word Server is sometimes referred to a physical server computer on a network. And the word Client, to represent a computer workstation on a user’s desk.
- These representations are also valid since the client workstation is making request from a server machine, but can be quite misleading.
- It’s important to understand that the word Client and Server as referred in a Client/Server Architecture, represent individual running processes or executables. Where the Client executable makes request to the Server executable, and the server executable processes and return the answer to the request to the client.
- What this means is that we can have a Client process run in the same physical machine as the Server process. The physical Hardware System or Computer has nothing to do with the location of the Client and Server process.
- So don’t get the process called SERVER confused with the computer called a server or the workstation called client and the process called CLIENT in the Client/Server architecture.

Client/Server and Database or DBMS

- IMPORTANT! It turns out that 95% of the time, the SERVER part of the Client/Server architecture is a Database Management System (DBMS) or some sort of Database application.
1.2.2 Client/Server Operating Systems

- The Client/Server architecture is so popular and used throughout the field of Computer Science, that it is also used to implement Network Operating Systems.
- For example, most of today’s popular networks Operating System Architectures are in reality Client/Server.
- Take for example a popular network operating systems architecture, such as Microsoft Windows NT/2000/2003.
- In the Microsoft Network Operating System architectures, the word Workstation (Client) and Server as used and most computer professionals refer to the workstation as the computer used by the user and the server as the backend computer. But in reality the computers have nothing to do with it. In fact what really makes a computer the client or workstation and the server is not the machine itself but the Operating systems running on these machines.
- Let’s look at the architecture in detail:

  ▪ Workstations (Client) Computers & Client Operating Systems - In this architecture, the word workstation or client is referred to the computer being used by a user. But the truth is as follows:
    - These computers run a CLIENT or desktop Operating System (OS) such as Windows 2000 Professional, or Windows NT Workstation, Windows XP or Windows 98 etc.
    - It is these Operating Systems that are really clients’ process, because the OS is an Operating system which makes request from a Server Operating Systems, which we will discuss below.
    - Windows 2000/2003 Professional, Windows XP and Windows NT Workstation are client Operating Systems, thus the computer they run on are called Clients or Workstations.
    - So as you can see, the computer has nothing to do with it, it is the Client OS running on these machines.

  ▪ Server Computers & Server Operating Systems - In this architecture, the word server is referred to the computer in the backend that contain all the user’s data and other programs which can be distributed or used by many workstations. But the truth is as follows:
    - These computers run a SERVER Operating System (OS) such as Windows 2000 Server, Windows 2003 Server, or Windows NT Server.
    - These Operating Systems that are really Server Operating Systems, because the OS is an Operating system which manages or handles request from Client Operating Systems.
    - Windows 2000/2003 Server and Windows NT Server are server Operating Systems, thus the computer they run on are referred to as Servers.
    - So as you can see, the computer has nothing to do with it, it is the Server OS running on these machines.

- So, once again it’s important that you understand that the word Client and Server as referred in a Client/Server Architecture, represent individual running processes or executables, which can be Operating systems as well and not the physical computers.

Server Operating Systems & Database

- Well, guess what? It turns out that the Server Operating Systems contain a Database.
- This database is used to store all the objects which make up the network resources, such as Computers, Users, Groups, Global Policies, etc.
- In Windows 2000/20003 this Database is called: ACTIVE DIRECTORY
- The Active Directory is nothing more than a Database stored on Server Operating System that contains records with all the objects in the network and their relationship etc.
- This Database gets replicated among all the Servers, which run the Server Operating Systems so that they are all in Sync or CONSISTENT.
- Like I stated in the last section, the SERVER portion of a Client/Server is usually a Database. Well, even in Operating Systems the server contains some sort of Database Application.
1.2.3 Client/Server Architecture & Application Development

- OK, now let’s focus again on the Client/Server architecture when it applies to application development or how applications are implemented in the field of Computer Science and where does the Database fits in into this schema.
- The Client/Server are usually implemented as follows:
  - **Client** – Front-End process or executable program written in languages like Visual Basic.NET, C#, Java etc.
  - **Server** – Back-End process or executable program that services or complies with the request made by a client process. This is always 99% of the times a Database Management System (DBMS).

- At this point you should clearly see that what we have between these two entities is really a DATABASE APPLICATION!!!!!!

**Client/Server Architectures that use DBMS**

- The **Client/Server Architecture** is composed of four basic architectures. They are as follows:
  - Single-Tier
  - Two -Tier
  - Three-Tier
  - Web-Based

- In the next several sections we will look at them in detail
1.2.4 Desktop Database (Single-Tier Client/Server)

- Desktop databases are designed to run on desktop personal computers, unlike DBMS, which need to run on servers.
- Desktop databases, such as Microsoft Access, are simple and powerful. A user with average computer skills can quickly build a database to store and organize just about any type of information, then build queries to extract the data without knowing anything about the language used to write queries (SQL).
- Desktop databases are flexible and portable. Since they are composed of a database file (.mdb in Access). The database file contains the tables, queries, forms, reports, etc.
- The database file is stored on a hard drive; therefore, you can copy, move, or delete it just like any other file.
- If the database file is stored on the server, multiple PCs can load the database into their own workspace, thus sharing the data with other colleagues.
- The main point to keep in mind here is that this type of database runs on one machine only.
- Some characteristics of desktop databases are as follows:
  - **Limited Storage Capacity**: MS Access only allows you to store up to 2 gigabytes of data.
  - **Inexpensive**: Desktop databases are cheap and ideal if your budget is limited.
  - **Limited functionality**: Desktop databases don’t have the powerful functionality that DBMS have.
  - **Limited Concurrent Connections**: MS Access only allows up to 255 concurrent connections.
  - **Large Bandwidth requirements**: Entire file virtual copy of the database file is copied when loaded to client via network.

- Diagram below illustrates the Desktop Database, which is also a single-tier Client/Server application:

```
Desktop Database & Client/Server
- A Desktop Database is a single-tier Client/Server application
- Single-Tier means it runs on ONE machine. Don’t get this confused with both the CLIENT and SERVER process running in the same machine.
- What we mean here is that it actually runs on ONE machine because it is a SINGLE PROCESS. There is no two separate process or executable.
- The CLIENT & SERVER process are combined into ONE process, which naturally can only run in one machine.
- This is why Desktop Databases run on ONE machine, because they are only one process, the Front-End and Back-End database are combined into one executable. NO NETWORK PROCESSING HERE! ONLY CLIENT(WORKSTATION) PROCESSING!
```
**Desktop Databases and Network Share**

- As stated earlier, the database is actually a flat file. The *database file* is stored on a hard drive; therefore you can copy, move or delete it just like any other file.

- You can also store *database file* is stored on a server computer or any computer on a Network. This will enable the database file to be *Shared*. Therefore multiple PCs use the database thus sharing the data with other colleagues.

- **IMPORTANT!** Remember, that this type of database is a *Single-Tier Client/Server* and only ONE PROCESS. Just because we can copy the Database File onto a machine on the network, and the database shared by many users, DOES NOT means that this is Network Database or that the Database is RUNNING on the Server. NO, it is still Single-tier, what actually happens is that a *Virtual copy* or the entire *database file* is copied to each user’s computer sharing the database, thus every user really has the entire database file locally on their workstation.

- The diagram below illustrates this concept:
1.2.5 Database Servers (Two-Tier Client/Server)

- Database Servers or DBMS systems are more powerful than their desktop counterparts.
- Their storage mechanism is different and more efficient than the desktop database files.
- More important, they are separate EXECUTABLE or PROCESS that run on their own, unlike their Desktop counterpart.
- They are Distributed Systems, which mean they can server multiple number of users and can be placed in many servers and they can replicate so that their data is in Sync or Consistent.
- Examples of database servers are Oracle, Microsoft SQL Server & MSDE (New)
- The new MSDE or Microsoft SQL Desktop Engine is the same as SQL Server, but without the management Tools and limited to a smaller number of users and storage capacity. It is design to work like a Desktop Database but with True Client/Server features.
- With database servers, the user front-end applications do not communicate with the database storage directly, but through a service running in the background or the DBMS.
- DBMS have the following characteristics:
  - **Large Storage Capacity**: DBMS like MS SQL allows you to store up to a terabyte of data.
  - **Expensive**: DBMS are expensive and require an expensive server in order to run.
  - **Large Concurrent Connections**: DBMS can allow from 5,000 to 10,000 concurrent connections.
  - **Greater functionality**: DBMS provide powerful functionality and work with server operating system to provide a more powerful and robust database system.
  - **Security**: Database servers offer advanced security since database access requires a login to the server, and additional authentication can be imposed on individual databases within the server.
  - **Less Bandwidth requirements**: Only the result of the query is passed through the network not the entire database file.

Database Servers and Two-Tier Client/Server

- The Database Server or DBMS system plays the most important role in a two-tier client server.
- Client/Server has become the standard for writing database applications. The Client/Server architecture consist of a client or front-end application, makes request to a Server application such as a Database or DBMS system, and the Server application handles the request or serves the client.
- The SERVER portion of this architecture is a DATABASE server. Whether the setup or configuration is, the server is usually the Database server or DBMS. The components of the architecture is as follows:
  1. **The Client** - The front-end (GUI) or client code.
  2. **The Server** - DBMS for Database Server

- The diagram below illustrates this architecture:
In the past we have discussed the two types of Two-Tier Client/Server which are Fat-Client/Thin-Server or Thin-Client/Fat-Server:

1. **Fat-Client/Fat-Server:** Client does most of the processing. User Interface and all data processing or *Business logic* are done on client. The Server is simply a DBMS managing the data or Database.
   - In this scenario the Client contains a lot of processing and is written in Visual Basics.NET or C#, Java etc to implement all the processing logic etc.
   - This scenario requires more powerful workstations and can have poor network performance since there is a lot of interaction between the data processing on the client and lots of request to the DBMS for data to process.

2. **Thin-Client/Fat-Server:** Client has less processing, usually User-Interface coding and very little processing. The Server on the other hand performs the Business Processing and complex processing.
   - In this scenario the Client contains a lot of processing and is written in Visual Basics.NET or C#, Java etc to implement all the processing logic etc.
   - This scenario requires less powerful workstations and can have better network performance since there most of the processing is done on the SERVER, only the results of the processing is returned to the client. Therefore there is less Data access interaction between the Client and the DBMS Server, thus better network performance.
   - Remember that the DBMS is only a Database Management System, so where does the DBMS get the power to perform the processing?
     1. Remember that the DBMS contains *Store Procedures, Triggers* and *custom-native language* which enable us to perform complex processing inside the DBMS. This was the popular method of *server-side* processing for many years and still used today.
     2. The new and latest trend is *Data-Access Business Objects.* Or Custom Business Classes created in Visual Basic.NET, C#, Java etc. that will run on the same or different machine as the DBMS, but performs the Data Access or Complex processing for the client.

Which of these two methods to use is still debated and both have their advantages and disadvantages. The Business Object approach is becoming more popular. But the Store Procedures approach is still used more frequently.

Developers are now using a combination of Both, which I feel is the best approach.
Workgroup & Departmental Databases

- A workgroup is a small team of users collaborating on the same project, application etc. Usually less than 25 users.
- A department is a functional unit within and organization; examples are Marketing, Human Resources, Accounting, and Finance etc. Departments are usually between 25 & 100 persons.
- Workgroup & Departmental Databases are stored in DBMS Database Servers. This Database Server is connected via the network to the workgroup desktops or clients.
- What distinguish a Workgroup Database from a Departmental Database is the physical attributes of the Server computer such as Disk Size, Processing power, Memory etc. Since a Departmental Database must accommodate more users, therefore the server computer should be more powerful. In addition, security issues are more crucial for a department than a workgroup. Also design requirements will be different, as well as the level of skill required for the administrators.
- The diagram bellows illustrates the Client/Server Workgroup and Department Databases:
Enterprise Databases (ERP & Data Warehousing)

- An enterprise database has a scope of the entire organization (many departments). Such databases are designed to support organization wide operations and decision making.
- In reality enterprise databases are a collection of independent departmental databases. A central management location can gather data from all databases etc.
- The evolution of Enterprise Databases have resulted in two major development:
  
  - **Enterprise Resource Management (ERP)** – A Business management software that integrates all functions of the enterprise, such as manufacturing, sales, finance, marketing, inventory, accounting and human resources.
    
    - ERP Systems are heavy dependents of databases. It is actually a very sophisticated database application.
    - ERP systems provide a single, consolidated view or the entire organization.
    - ERP Systems work and manage current operational data for an organization.
    - ERP systems are popular in Human Resource departments.
    - These applications are also finding their ways in Health-care organizations as well.
  
  - **Data Warehousing** – A large database which collects contents from all the various operational databases including desktop, workgroup, departmental and ERP databases. What is important of Data warehousing databases is that they allow users to work with historical data.
    
    - Data warehousing work and manage historical data for an organization.
    - Data warehousing applications are important in Health-care organizations where historical data for patients as well as the organization is required.

- The following illustration shows an example of an Enterprise Database being used in a Health Care Organization.
1.2.6 Web-Based Client/Server, the Internet and Database Applications

- The current most popular client/server architecture is the Internet or World Wide Web client/server.
- The popularity of the Internet has led to a new form of programming known as **Web Development**.
- Once again, the database server plays one of the most important roles and that is the storage and management of data.
- Using Web Development architecture, Internet database applications are written as follows:
  3) **The Client** - The front-end (GUI) or client is in written in HTML
  4) **The Server** - A **Web Server**, such as the Internet Information Server (IIS), is used to service the client's HTML requests. On the Web Server, complex processing is done using **Active Server Pages (ASP)** via VBScript language. The ASP code can be used to connect to another server or DBMS where the data is stored.
  5) **The Server** - DBMS or Database Server

1.2.7 Intranet & Web Development for Internal Database Applications

- This Internet Web Development standard is now being adapted for developing in-house business application or **Intranet** application.
- The difference between the Internet & Intranet architectures is that in the Internet architecture, the WEB Servers are located external to the company or in the World-Wide-Web, where in the Intranet, the servers are located internal to the company:
Chapter 2. Components or a Relational DBMS

2.1 Components of a Relational DBMS

2.1.1 The Components that make up a Database Management System

- In this course, we have no time to go into the details of database systems & design, but let's review the basics.
- A DBMS is primarily composed of the following basic components:
  - Tables
  - Queries (SQL)
    - Triggers
    - Stored Procedures
  - Reports

2.2 Relational Tables

2.2.1 Table Basics

- A Table is storage facility used by a relational database to store the data.
- Tables are the basic building blocks of a relational database.
  - A Table is composed of columns called fields & rows called records

**Fields** are the columns
**Records** are the rows

<table>
<thead>
<tr>
<th>Field 1</th>
<th>Field 2</th>
<th>Field 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data Type:**
- The data type defines the type of data to be entered in each field or column of the table. Data such as text, number or date etc.
- Each data type has the following characteristics:
  - **Properties:** Define the characteristics of a field such as how many characters can fit in the field, whether the field can be null or blank
  - **Data Type:** The type of data allowed to be entered in the column or field.
  - For example Microsoft Access provides the following data types:
<table>
<thead>
<tr>
<th>Data types</th>
<th>Description</th>
<th>Maximum Size</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Text or combination of words and numbers (numbers not used in calculations)</td>
<td>255 characters</td>
<td>none</td>
</tr>
<tr>
<td>Memo</td>
<td>Text that is too long to be stored in Text field.</td>
<td>65,535 characters</td>
<td>none</td>
</tr>
</tbody>
</table>
| Number     | Numeric numbers. You can choose from a variety of number type or properties such as:  
- **Integer**: whole numbers with no decimal points (size: -32K to +32K)  
- **Long Integer**: larger whole numbers (size: -2M to + 2M)  
- **Double**: Large Decimal point numbers (15 decimal places)  
- **Single**: Smaller Decimal point number (7 decimal places)  
- **Byte**: Small integer from 0 to 255  
- **Others**. | Depend on properties | Integer, Long Integer, byte, Currency, Single, Double, etc. |
| Date/Time  | Date/Time are stored in a special format such as:  
- **General Date**: (06/19/1994 5:34:23 PM)  
- **Long Date**: (Sunday, June 19, 1994)  
- **Short Date**: (06/19/1994)  
- **Long Time**: (5:34:23 PM)  
- **Short Time**: (17:34) | | |
| Currency   | Exactly like Double Number type, but formatted to display up to 4 decimal places and $ symbol. | | |
| AutoNumber | A numeric (long Integer) sequential number that is automatically incremented by one when ever a new record is added to the table | | |
| Yes/No     | A logical or Boolean value that can have the following formats:  
- **True/False**  
- **Yes/No**  
- **On/Off** | | |
| OLE Object | An Object that are linked and embedded to the field, such as a graphical picture, sound, document or Excel Spreadsheet. | | |
| Hyperlink  | Hyperlink entries that are path to an object, document, Web Page or other destinations | | |
2.2.2 Table Relationship

- The power of relational databases is their ability to store data with minimal duplication, and to link or relate data from different tables.
- The link between two tables is known as the **relationship**.
- Linking tables properly is important in database design in order to avoid entering duplicate data transactions. In order to link tables properly, we need to understand the concept of the **Primary Key**.

### Primary key

- A **Primary Key (PK)** is a field or column, which contains unique values and uniquely identifies each record.
- Most tables have at least one field that is selected as a primary key.
  - Examples of primary keys can be a social security number, a unique ID number, a driver license number, etc.
  - If a column is defined as the primary key, this will guarantee that every record on the table is unique due to the **PK**.
  - Primary keys are useful in eliminating duplicate records, since every record must be unique, entering a duplicate will violate the primary key.
  - Primary keys help control the order in which records are displayed, which makes it faster to find records.
- Primary keys are usually not mandatory, but are highly recommended in database design in order to maintain data integrity.

### CUSTOMER TABLE

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Harris</td>
<td>John</td>
<td>12/12/65</td>
<td>Brooklyn, NY</td>
<td>M</td>
</tr>
<tr>
<td>002</td>
<td>Flores</td>
<td>Gloria</td>
<td>1/23/70</td>
<td>New York, NY</td>
<td>F</td>
</tr>
<tr>
<td>003</td>
<td>Smith</td>
<td>Charles</td>
<td>6/16/68</td>
<td>Queens, NY</td>
<td>M</td>
</tr>
<tr>
<td>004</td>
<td>Chin</td>
<td>Lisa</td>
<td>3/8/73</td>
<td>Corona, NY</td>
<td>F</td>
</tr>
</tbody>
</table>

**Primary Key Field**

### Foreign Key

- The **Foreign Key (FK)** is the field or combination of fields, linked to the **Primary Key** field of another table. In other words when the primary key field of a table, exist in another table, this field is known as the **Foreign Key**.

### CUSTOMER TABLE

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>Flores</td>
<td>Gloria</td>
<td>1/23/70</td>
<td>New York, NY</td>
<td>F</td>
</tr>
<tr>
<td>003</td>
<td>Smith</td>
<td>Charles</td>
<td>6/16/68</td>
<td>Queens, NY</td>
<td>M</td>
</tr>
<tr>
<td>004</td>
<td>Chin</td>
<td>Lisa</td>
<td>3/8/73</td>
<td>Corona, NY</td>
<td>F</td>
</tr>
</tbody>
</table>

**Primary Key Field**

### VIDEO-TRANSACTION TABLE

<table>
<thead>
<tr>
<th>TAPE_ID</th>
<th>CUST_ID</th>
<th>RENTAL_DATE</th>
<th>RETURN_DATE</th>
<th>TRANS_CLOSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>002</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
<td>TRUE</td>
</tr>
<tr>
<td>103</td>
<td>001</td>
<td>01/01/2001</td>
<td>01/03/2001</td>
<td>TRUE</td>
</tr>
<tr>
<td>101</td>
<td>003</td>
<td>01/03/2001</td>
<td>01/03/2001</td>
<td>TRUE</td>
</tr>
<tr>
<td>102</td>
<td>001</td>
<td>01/01/2001</td>
<td>01/04/2001</td>
<td>FALSE</td>
</tr>
<tr>
<td>104</td>
<td>004</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

**Foreign Key Field**

- **Primary Keys and Foreign Keys** do not have to be one column but can be a combination of columns that make up the key.
Relationships

- Relationships tell you how the data of one table relates or is linked to another table. In reality a table by itself is useless if you cannot related to other data stored in other tables.
- There are basic two types of relationships:
  - **One-to-One**: In the first table, each value in the Primary Key field is unique and in the second table, each value in the Foreign Key field is also unique. This means that the foreign key of the second table is the primary key to that table as well.
  - In other words, *if a primary key of one table is linked to a single field primary key of another table a one-to-one relationship exists.*

![Table with Primary Key Field](STUDENT TABLE)

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
</tbody>
</table>

![Table with Foreign Key Field](TRANSACTION TABLE)

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
</tbody>
</table>

- **One-to-Many**: In the first table, each value in the Primary Key field is unique and in the second table, each value in the Foreign Key field is NOT unique. There can be more than one value in the foreign key field that matches the primary field, therefore the foreign key field of the second table is NOT a primary key to that table.
- In other words, *if a primary key of one table is linked to a field on a second table that is NOT a primary key, a one-to-many relationship exists.*

![Table with Primary Key Field](STUDENT TABLE)

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
</tbody>
</table>

![Table with Foreign Key Field](SID)

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
</tbody>
</table>

Referential Integrity

- By defining relationships between tables, we are telling the database which Foreign Keys relate to which Primary Keys.
- Once we have done this, we can turn on a feature of a database known as **Referential Integrity**.
- With **Referential Integrity** enforced, we ensure that data in related tables remain consistent, no duplicate records exist and correct data is entered.
- **Referential Integrity** enforces that when populating a record of a table with a foreign key, the primary key of the linked table must have a matching value, therefore all tables are consistent.
- For example, in the relationship between the STUDENT TABLE and THE CLASS-TAKEN TABLE, with **Referential Integrity**, if you attempt to add a new record onto the CLASS-TAKEN TABLE, which does not exist in the STUDENT TABLE, and error will occur and the database will inform you that **Referential Integrity** is being violated.
2.3 Introduction to Queries, Server-Side Processing, View and Reports

2.3.1 Introduction

- Queries are requests that you send to the database.
- Queries are used to retrieve, insert, update or delete data in any given database table.
- The **Structure Query Language (SQL)** is the standard database query language recognized by every DBMS system.
- Note that there are many variations of SQL depending on which DBMS system used, for example Oracle uses **PL/SQL** while Microsoft SQL Server uses **Transact SQL**.
- A single SQL statement applied to a table or database can retrieve a specific record or a number of records quickly and easily, and can perform arithmetic operation on that data.
- There are four basic SQL statements design to handle data:

  1) **SELECT** statement - used to retrieve records from a table
  2) **DELETE** statement - used to remove records from a table
  3) **INSERT** statement - used to insert records into a table
  4) **UPDATE** statement - used to modify records in a table

- Example of an SQL statement:

  ```sql
  SELECT Last Name, First Name
  FROM Customer
  ```

2.4.1 Searching the Database

- Because most databases contain hundreds or thousands of records, the capability of the program to find and display quickly an individual record or a group of record is essential.
- There are basically two ways to search or find records:

  1) **Query to Retrieve Data with no criteria**: The first situation occurs when you need to locate a record or records in the database with no specific criteria. For example:

  ```sql
  SELECT Last Name, First Name
  FROM Customer
  ```

  2) **Query or Search for a record based on criteria**: The second situation occurs when you need to answer a query or question about the data in the database depending on a criteria or condition. For example:

  ```sql
  SELECT Last Name, First Name
  FROM Customer
  WHERE Customer_ID = 001
  ```

2.3.2 Advanced Server-Side Processing SQL Functionality

- In almost if not all DBMS there are advanced SQL server-side functionality available. We are not going to go into great details on how they work, but we need to understand their concepts.
- This Server-side functionality allows the server to handle more complex processing and thus alleviating the client or front-end GUI from handling such processing.
- This Server-side processing allows for a client/server concept known as **thin-client & fat server** model. The following sections will discuss these server-side processing features.
Triggers
- Triggers are a collection of SQL statement-like functions that automatically execute when an INSERT, DELETE or UPDATE statement is applied to a table.
- The key words here are SQL-like, functions & automatically execute.
- Triggers are basically functions you create, composed of SQL statements & non-SQL statements. These NON-SQL statements are usually specific to a particular DBMS.
- You can then assign a trigger function to a table and when an INSERT, DELETE, or UPDATE statement is applied to the table, an event is fired (triggered), thus executing the block of code.
- Therefore triggers are just automatic events that are executed when a modification is made to a table.
- Since these internal triggers reside on the DBMS, they allow the server to take the responsibility for the Business Logic.
- Therefore triggers allow us to enforce business rules, ensure data integrity and automate tasks without the client interaction, thus reducing Network Traffic.

Stored Procedures
- Stored Procedures are pre-compiled SQL statements procedures that accept arguments from the client (GUI).
- The Stored Procedures can be called thus executed from a client or trigger.
- Like Triggers, DBMS provide a language that extends the SQL statements used Stored Procedures. These NON-SQL statements are usually specific to a particular DBMS.
- Store Procedures are used to automate many of the complex processing on the server thus allowing the server to take the responsibility for the Business Logic.
- Stored Procedures like Triggers allows us to enforce business rules, and automate tasks without the client interaction, thus reducing Network Traffic.
- Store Procedures allow us to write powerful & complex SQL statements on the server and provide the following advantage:
  - Store and execute complex operations on the server, therefore we keep the client of front-end light. SQL statements written on the client can be kept small, so much so that they can be simply composed of the call to the Stored Procedure. And it is the Stored Procedure on the sever side that does all the work.
  - They are maintained on the server side, therefore they can be modified quickly.
  - Stored procedures can be used to automate tasks, such as purging data, archiving data or creating tables on the fly.
  - Stored procedures run much faster than standard SQL statements since they are pre-compiled at the database level.
  - Stored Procedures reduces Network Traffic by allowing the clients to send smaller SQL statements.
  - Stored Procedures can be used to handle errors at the database level and not at the client.
- Disadvantages to Store Procedures are:
  - May become difficult to program maintain & debug.
  - Programming syntax can be difficult to learn
- Normally the advantages of stored procedures out-weight the disadvantages. Store Procedures is very popular and recommended for most business applications since the some of the data processing is done on the server.

2.3.3 Views
- Views are SQL statements that are stored on the server. These views create virtual tables that allow clients to retrieve and work with the virtual table's data. View can be used to retrieve specific data from one or more tables, later if a client needs data from those same tables, the client can apply an SQL statement against the view thus keeping the SQL statement simple.
- Therefore you can write a complex SQL statements and define it as a view. The client that requires data from the view can then execute a short SQL statement to retrieve that data again and again.
- Views can be used to add security to a table by simply allowing client access to a view of a table or tables rather than the tables themselves. This way the client only has access to the portion of the table you want them to have.

2.3.4 Reports
- Reports are printed and formatted presentation of specified records. Reports are used to summarize and print data.
- Reports allow you to customize the appearance of the data. A report can be an Invoice, Monthly summary report etc.
- In a bank database, reports are printed monthly and sent to customers. Some DBMS provide a GUI like interface to help you create the format for reports. Note that the data generated by reports come from SQL statements, Store Procedures etc.
Chapter 3 Building SQL Queries

3.1 Building SQL Statements (Introduction)

3.1.1 Introduction

- The Structure Query Language (SQL) is the standard database query language used to retrieve, insert, update or delete data in any given database table.
- There are four basic SQL statements design to handle data:
  1) SELECT statement - used to retrieve records from a table
  2) DELETE statement - used to remove records from a table
  3) INSERT statement - used to insert records into a table
  4) UPDATE statement - used to modify records in a table

- Note that when creating queries that retrieve data from the database tables that the results of a query is also a table.
- In the following sections we will be reviewing the syntax & statements for creating SQL queries. For teaching purpose, we will be using the table design or schema shown below in most of our query examples. Note the relationship between the tables and how Referential Integrity is maintained due to the Primary & Foreign Keys of the tables:
### Customer Table

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Harris</td>
<td>John</td>
<td>12/12/1968</td>
<td>129 East 34 St Brooklyn, NY 11210</td>
<td>M</td>
<td>718-555-1232</td>
</tr>
<tr>
<td>3333</td>
<td>Smith</td>
<td>Charles</td>
<td>05/15/1968</td>
<td>139 Queen Blvd, Flushing, NY 11233</td>
<td>M</td>
<td>718-555-1234</td>
</tr>
<tr>
<td>4444</td>
<td>Chin</td>
<td>Yin</td>
<td>03/08/1972</td>
<td>2514 Flatbush Ave, Brooklyn, NY 11224</td>
<td>F</td>
<td>718-555-6758</td>
</tr>
<tr>
<td>5555</td>
<td>Rivera</td>
<td>Nora</td>
<td>03/23/1965</td>
<td>342 Woodhaven Blvd, Woodhaven NY 11234</td>
<td>F</td>
<td>718-555-9007</td>
</tr>
<tr>
<td>6666</td>
<td>Rivera</td>
<td>Charles</td>
<td>01/22/1968</td>
<td>123 Smith St, Brooklyn NY 11219</td>
<td>M</td>
<td>718-444-2342</td>
</tr>
<tr>
<td>7777</td>
<td>Clinton</td>
<td>George</td>
<td>11/14/1967</td>
<td>134 President's blvd, Westchester NY 10023</td>
<td>M</td>
<td>718-555-6566</td>
</tr>
<tr>
<td>8888</td>
<td>Clinton</td>
<td>Hillary</td>
<td>01/22/1958</td>
<td>134 President's blvd, Westchester NY 10023</td>
<td>M</td>
<td>718-555-5555</td>
</tr>
</tbody>
</table>

### VideoTape Table

<table>
<thead>
<tr>
<th>TapeID</th>
<th>Title</th>
<th>Category</th>
<th>TapeStatus</th>
<th>Rating</th>
<th>DailyRentalCost</th>
<th>DailyLateFee</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Terminator</td>
<td>Adventure</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>102</td>
<td>Star Wars</td>
<td>Sci-Fi</td>
<td>F</td>
<td>PG</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>103</td>
<td>The Matrix</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>104</td>
<td>Forrest Gump</td>
<td>Family</td>
<td>T</td>
<td>PG-13</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>105</td>
<td>Scary Movie</td>
<td>Comedy</td>
<td>F</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>106</td>
<td>Easy Ride</td>
<td>Adult</td>
<td>T</td>
<td>X</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>107</td>
<td>Terminator</td>
<td>Adventure</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>108</td>
<td>The Matrix</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>109</td>
<td>Star Wars</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
</tbody>
</table>

### Transaction Table

<table>
<thead>
<tr>
<th>Transaction_ID</th>
<th>Customer_ID</th>
<th>Tape_ID</th>
<th>RentalDate</th>
<th>ReturnDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1111</td>
<td>101</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>2</td>
<td>3333</td>
<td>102</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>3</td>
<td>2222</td>
<td>105</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>4</td>
<td>1111</td>
<td>103</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>104</td>
<td>01/01/2001</td>
<td>01/03/2001</td>
</tr>
<tr>
<td>*</td>
<td>(AutoNumber)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
3.2 SELECT Statement & Associated Clauses

3.2.1 SELECT Statement

- The SELECT statement is used to retrieve records from tables.
- The standard SELECT statement syntax is as follows:

```
SELECT <Column list>
FROM <Table list>
WHERE <Criteria condition>
```

- The following table list the clauses that can be attached to the SELECT statement to create more sophisticated queries:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Placement</th>
<th>Behavior &amp; Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Required</td>
<td>Identifies the columns to be returned</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Optional</td>
<td>Specifies that each record returned from the query should be unique</td>
</tr>
<tr>
<td>FROM</td>
<td>Required</td>
<td>Identifies the tables that the query will affect</td>
</tr>
<tr>
<td>WHERE</td>
<td>Optional</td>
<td>Identifies the criteria that the records returned must satisfy</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Optional</td>
<td>Groups records, based on the values in the grouping columns</td>
</tr>
<tr>
<td>HAVING</td>
<td>Optional</td>
<td>Allows filtering out of the group that don't satisfy a search condition</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Optional</td>
<td>Identifies the order of sorting of columns values for a particular field or column</td>
</tr>
</tbody>
</table>

SELECT Statement Column List

Returning All or Restricted Field Query Results

- In the column list you have the option of selecting ALL the fields from a table as follows:

```
SELECT *
FROM CUSTOMER
```

Query Results:
- A table listing all the fields of the Customer table. In affect the entire Customer table is being returned.

- In the column list, you can restrict the number of fields returned by listing each field in the column list:

```
SELECT LastName, FirstName, BirthDate
FROM CUSTOMER
```

Query Results:
- A table listing only the LastName, FirstName & BirthDate fields of the Customer table.
Returning DISTINCT Values

In the column list you can use the DISTINCT clause to specify that on a particular field, the records returned from the query must be unique or distinct, in other words "no duplicate records" on a field:

```
SELECT DISTINCT LastName
FROM CUSTOMER
```

Query Results:
- A table listing only unique records based on the LastName field.

<table>
<thead>
<tr>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Clinton</td>
</tr>
<tr>
<td>Flores</td>
</tr>
<tr>
<td>Harris</td>
</tr>
<tr>
<td>Rivera</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>

Creating Alias Named for Fields

In the column list you can use the AS Keyword to allow us to rename the resultant fields of a query:

```
SELECT TapeStatus AS TapeAvailable
FROM VIDEOTAPE
```

Query Results:
- A table with column renamed:

<table>
<thead>
<tr>
<th>TapeAvailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>

Concatenating Multiple Field:

You can also concatenate multiple fields:

```
SELECT LastName + ',' + FirstName AS Name
FROM CUSTOMER
```

Query Results:
- A table with columns concatenated and renamed:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith,John</td>
</tr>
<tr>
<td>Flores,Gloria</td>
</tr>
<tr>
<td>Smith,Charles</td>
</tr>
<tr>
<td>Chin,Yin</td>
</tr>
<tr>
<td>Rivers,Nora</td>
</tr>
<tr>
<td>Rivera,Charles</td>
</tr>
<tr>
<td>Clinton,George</td>
</tr>
<tr>
<td>Clinton,Hillary</td>
</tr>
</tbody>
</table>
### 3.2.2 WHERE Clause

- The WHERE clause gives you the ability to retrieve records based on highly specific criteria. The WHERE clause is the most common method used to restrict or filter the data returned by a SQL query.
- The syntax is as follows:

```sql
SELECT * 
FROM CUSTOMER 
WHERE LastName = 'Clinton'
```

#### Query Results:
- All records whose last name is Clinton.

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>7777</td>
<td>Clinton</td>
<td>George</td>
<td>11/14/1955</td>
<td>134 President's bld, Westchester NY 10023</td>
<td>M</td>
<td>718-555-5555</td>
</tr>
<tr>
<td>8888</td>
<td>Clinton</td>
<td>Hillary</td>
<td>01/22/1956</td>
<td>134 President's bld, Westchester NY 10023</td>
<td>F</td>
<td>718-555-5555</td>
</tr>
</tbody>
</table>

### Arithmetic Comparison & Boolean Operator in WHERE Clause

- You can use the WHERE clause with the comparison operator in the following table:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

```sql
SELECT * 
FROM CUSTOMER 
WHERE Customer_ID > 4444
```

#### Query Results:
- A table listing all customers with ID > 4444:

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>7777</td>
<td>Rivera</td>
<td>Nora</td>
<td>03/23/1966</td>
<td>342 Woodhaven Blvd, Woodhaven NY 11234</td>
<td>F</td>
<td>718-456-5678</td>
</tr>
<tr>
<td>6666</td>
<td>Rivera</td>
<td>Charles</td>
<td>01/22/1966</td>
<td>123 Smith St, Brooklyn NY 11219</td>
<td>M</td>
<td>718-444-2342</td>
</tr>
<tr>
<td>7777</td>
<td>Clinton</td>
<td>George</td>
<td>11/14/1955</td>
<td>134 President's bld, Westchester NY 10023</td>
<td>M</td>
<td>718-555-5555</td>
</tr>
<tr>
<td>8888</td>
<td>Clinton</td>
<td>Hillary</td>
<td>01/22/1956</td>
<td>134 President's bld, Westchester NY 10023</td>
<td>F</td>
<td>718-555-5555</td>
</tr>
</tbody>
</table>
You can use the WHERE clause with the Logical operator in the following table:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Expr1 AND Expr2</td>
<td>Returns TRUE only and only if both expressions are TRUE</td>
</tr>
<tr>
<td>OR</td>
<td>Expr1 OR Expr2</td>
<td>Returns TRUE if Expr1 or Expr2 (or both) are TRUE</td>
</tr>
<tr>
<td>NOT</td>
<td>NOT Expr</td>
<td>Returns the inverse of the Expression. Returns TRUE if Expr is FALSE and FALSE if the Expr is TRUE</td>
</tr>
</tbody>
</table>

```
SELECT * FROM CUSTOMER WHERE Customer_ID > 4444 AND Gender = 'F'
```

Query Results:
- A table listing customers with ID > 4444 and female:

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone_Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8448</td>
<td>Rivera</td>
<td>Nora</td>
<td>03/23/1965</td>
<td>342 Woodhaven Blvd, Woodhaven NY 11234</td>
<td>F</td>
<td>716-555-9067</td>
</tr>
<tr>
<td>6366</td>
<td>Clinton</td>
<td>Hillary</td>
<td>01/22/1955</td>
<td>134 President's Blvd, Westchester NY 10023</td>
<td>F</td>
<td>716-555-5555</td>
</tr>
</tbody>
</table>

The IN, LIKE & BETWEEN Predicates in the WHERE Clause

- The IN predicate is an alternative to the Logical OR operator. It makes the SQL statement shorter and less intimidating specially when you have multiple values to locate:

```
SELECT * FROM VIDEOTAPE WHERE Category IN ('Sci-Fi', 'Adventure')
```

Query Results:
- A table listing all columns where category is equal to Sci-Fi OR Adventure:

<table>
<thead>
<tr>
<th>TapeID</th>
<th>Title</th>
<th>Category</th>
<th>TapeStatus</th>
<th>Rating</th>
<th>DailyRentalCo</th>
<th>DailyLateFee</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Terminator</td>
<td>Adventure</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>102</td>
<td>Star Wars</td>
<td>Sci-Fi</td>
<td>F</td>
<td>PG</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>103</td>
<td>The Matrix</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>107</td>
<td>Terminator</td>
<td>Adventure</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>108</td>
<td>The Matrix</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
<tr>
<td>109</td>
<td>Star Wars</td>
<td>Sci-Fi</td>
<td>T</td>
<td>R</td>
<td>$2.00</td>
<td>$1.00</td>
</tr>
</tbody>
</table>
The LIKE allows you to specify a pattern to search for when implementing a SQL statement. It can be used to locate strings in the beginning, middle or end of a field. It works in conjunction with Wild Characters such as the % for Microsoft SQL or * for Microsoft Access:

```
SELECT * FROM CUSTOMER WHERE Address LIKE "*Brook*"
```

Query Results:
- A table listing all columns where the Address field contains a string with the letters Brook:

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone Numbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Harris</td>
<td>John</td>
<td>12/12/1965</td>
<td>123 East 34 St Brooklyn, NY 11210</td>
<td>M</td>
<td>718-555-1232</td>
</tr>
<tr>
<td>4444</td>
<td>Chin</td>
<td>Yin</td>
<td>03/08/1972</td>
<td>2514 Flatbush Ave, Brooklyn, NY 11224</td>
<td>F</td>
<td>718-555-6758</td>
</tr>
<tr>
<td>6666</td>
<td>Rivera</td>
<td>Charles</td>
<td>01/22/1968</td>
<td>123 Smith St, Brooklyn NY 11219</td>
<td>M</td>
<td>718-444-2342</td>
</tr>
</tbody>
</table>

The BETWEEN allows you to specify a range of possible values, similar to the comparison operators. It is an alternative to using many comparison operators. The BETWEEN operator goes hand and hand with the AND operator:

```
SELECT * FROM CUSTOMER WHERE Customer_ID BETWEEN 2222 AND 4455
```

Query Results:
- A table listing all columns where the Customer_ID field range is >= 2222 and <=4455:

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>LastName</th>
<th>FirstName</th>
<th>BirthDate</th>
<th>Address</th>
<th>Gender</th>
<th>Phone Numbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>3333</td>
<td>Smith</td>
<td>Charles</td>
<td>05/16/1968</td>
<td>139 Queen Blvd, Flushing, NY 11233</td>
<td>M</td>
<td>718-555-1234</td>
</tr>
<tr>
<td>4444</td>
<td>Chin</td>
<td>Yin</td>
<td>03/08/1972</td>
<td>2514 Flatbush Ave, Brooklyn, NY 11224</td>
<td>F</td>
<td>718-555-6758</td>
</tr>
</tbody>
</table>

Working with Dates in the WHERE Clause
- Each DBMS system has as specific rules for working with dates. Typically, dates will be enclosed either by # (Access) or ’ (SQL Server).
- Once enclosed in # or ’, then the date can be manipulated as if it were a string:

```
SELECT * FROM TRANSACTION WHERE ReturnDate BETWEEN #01/01/2001# AND #01/03/2001#
```

Query Results:
- A table listing all columns where ReturnDate is between the date specified in the query:

<table>
<thead>
<tr>
<th>Transaction_ID</th>
<th>Customer_ID</th>
<th>Tape_ID</th>
<th>RentalDate</th>
<th>ReturnDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1111</td>
<td>101</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>2</td>
<td>3333</td>
<td>102</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>4</td>
<td>1111</td>
<td>103</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>104</td>
<td>01/01/2001</td>
<td>01/03/2001</td>
</tr>
</tbody>
</table>
Working with NULLS in the WHERE Clause

- The NULL predicate can be used to find records where the value in a selected field is null.
- The NULL predicate works in conjunction with the IS and the NOT keyword.
- Note that a null value occurs when a field has not yet been assigned. It is not the same as a space character.
- Example:

```
SELECT * 
FROM   CUSTOMER
WHERE  PhoneNumber IS NULL
```

Query Results:
- A table listing all columns and returning the records where the PhoneNumber is null or not been assigned as of yet.

- Example:

```
SELECT * 
FROM   CUSTOMER
WHERE  PhoneNumber IS NOT NULL
```

Query Results:
- A table listing all columns and returning the records where the PhoneNumber is NOT null or has been assigned values.

3.2.3 The ORDER BY Clause

- The ORDER BY clause when specified to a specific field or fields will allow you to sort the records that are returned from a query into ascending or descending order.
- The syntax is as follows:

```
SELECT FirstName, LastName 
FROM CUSTOMER 
WHERE Gender = 'F' 
ORDER BY LastName
```

Query Results:
- Returns listing of First Names & LastName records who are Females and results are ordered by Last Name.

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prin</td>
<td>Chin</td>
</tr>
<tr>
<td>Hillary</td>
<td>Clifton</td>
</tr>
<tr>
<td>Gloria</td>
<td>Flores</td>
</tr>
<tr>
<td>Nora</td>
<td>Rivera</td>
</tr>
</tbody>
</table>
3.2.4 The GROUP BY Clause

- The GROUP BY allows you to group records based on a field.
- Note that the records are grouped and kept in a temporary table, and the SELECT statement is applied to that temporary table and the results returned.
- The syntax is as follows:

```
SELECT FirstName, Gender
FROM CUSTOMER
GROUP BY Gender, FirstName
```

Query Results:
- Returns listing of First Names & LastName records who are Females and results are ordered by Last Name.

<table>
<thead>
<tr>
<th>FirstName</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stone</td>
<td>F</td>
</tr>
<tr>
<td>Hillary</td>
<td>F</td>
</tr>
<tr>
<td>Nora</td>
<td>F</td>
</tr>
<tr>
<td>Yin</td>
<td>F</td>
</tr>
<tr>
<td>Charles</td>
<td>M</td>
</tr>
<tr>
<td>George</td>
<td>M</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
</tr>
</tbody>
</table>

3.2.5 SQL Aggregated Functions

- The aggregate functions are built in mathematical functions that you can apply to queries.
- Note that the aggregated functions return a value and the resultant value is a table with a nameless column. That is the result is returned in a table with a column without a name. Therefore usually queries that use aggregated functions use the AS predicate to give the column a name.
- Note that you can use the Aggregated functions with the GROUP BY & HAVING clause which I leave to you to research and learn.
- The list of aggregated functions are as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>Returns the sum of all the values in a field or column</td>
</tr>
<tr>
<td>COUNT</td>
<td>Returns the count of all the values in a field based on a criteria</td>
</tr>
<tr>
<td>MIN</td>
<td>Returns the Minimum of all the values in a field or column</td>
</tr>
<tr>
<td>MAX</td>
<td>Returns the Maximum of all the values in a field or column</td>
</tr>
<tr>
<td>AVG</td>
<td>Returns the Average of all the values in a field or column</td>
</tr>
</tbody>
</table>

**The SUM Aggregated Function**

- Returns the sum of all the values in a field or column:

```
SELECT SUM (DailyRentalCost)
FROM VIDETAPE
```

Query Results:
- A table listing the sum of the values in the DailyRentalCost column. NOTE that the resulting column does NOT have a valid name.

| Exp1000 | 1800 |
Using the AS predicate to give the returned column a valid name:

```
SELECT SUM(DailyRentalCost) AS TotalRentalCost
FROM VIDETAPE
```

**Query Results:**
- A table listing the sum of the values in the DailyRentalCost column and the column labeled as TotalRentalCost.

<table>
<thead>
<tr>
<th>TotalRentalCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$18.00</td>
</tr>
</tbody>
</table>

**The COUNT Aggregated Function**

- Returns the count of all the values in a field based on a criteria:

```
SELECT COUNT(Customer_ID) AS NumberofCustomers
FROM CUSTOMER
```

**Query Results:**
- A table listing the number of customer in the customer table based on the customer column.

<table>
<thead>
<tr>
<th>NumberofCustomers</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

**The MIN & MAX Aggregated Function**

- `MIN`, returns the Minimum of all the values in a field or column
- `MAX`, returns the Maximum of all the values in a field or column

```
SELECT MIN(RentalDate) AS MinRentalDate, MAX(ReturnDate) AS MaxRentalDate
FROM TRANSACTION
```

**Query Results:**
- A table listing the minimum rental date & maximum return date.

<table>
<thead>
<tr>
<th>MinRentalDate</th>
<th>MaxReturnDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/2001</td>
<td>01/04/2001</td>
</tr>
</tbody>
</table>

**The AVG Aggregated Function**

- Returns the Average of all the values in a field or column:

```
SELECT AVG(DailyRentalCost) AS AverageRentalCost
FROM VIDETAPE
```

**Query Results:**
- A table listing the average value of the DailyRentalCost table.

<table>
<thead>
<tr>
<th>AverageRentalCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2.00</td>
</tr>
</tbody>
</table>
3.3 UPDATE Statement & Associated Clauses

3.3.1 UPDATE Statement

- The UPDATE statement allows you to modify records in tables.
- The UPDATE statement is used in conjunction with the SET clause can be used to change the data within a record(s)
- Syntax:

\[
\begin{align*}
\text{UPDATE} & \quad \text{<Table Name>} \\
\text{SET} & \quad \text{<Field Name>} = \text{<Value>} \\
\text{WHERE} & \quad \text{<Criteria condition>}
\end{align*}
\]

- Note that in the WHERE clause, the syntax, predicates, aggregated functions & clauses still apply as shown in previous section.
- Example:

```
UPDATE Customer
SET LastName = 'Bush'
WHERE Customer_ID = 8888
```

Query Results:
- The table Customer is modified by changing the last name of the ID 8888.

- After table is modify, we can use a query to see if the value was committed:

```
SELECT * 
FROM CUSTOMER
```

Query Results:
- A table listing all values of the Customer table. Note that the LastName of ID 8888 was changed to Bush.
3.4 INSERT Statement & Associated Clauses

3.4.1 INSERT Statement

- The INSERT statement allows you to add or insert a new record into a table(s).
- The INSERT statement is used in conjunction with the INTO and VALUES clause in order to insert a record.
- Syntax:

```
INSERT INTO <Table Name> ([Field Name 1], [Field Name 2], [Field Name 3], [Field Name n])
VALUES ([Field Data 1], [Field Data 2], [Field Data 3], [Field Data n])
```

- Note that the field names must be in the exact same order as they appear on the table.
- Example:

```
INSERT INTO Transaction (Transaction_ID, Customer_ID, Tape_ID, RentalDate, ReturnDate)
VALUES (6, 5555, 108, '#01/04/2001#', '#01/06/2001#')
```

Query Results:
- Add a record to the Transaction table.

- After table is modify, we can use a query to see if the value was committed:

```
SELECT * FROM TRANSACTION
```

Query Results:
- A table listing all values of the Transaction table. Note that a new record was added.
3.5 DELETE Statement & Associated Clauses

3.5.1 DELETE Statement

- The DELETE statement allows you to remove or delete a record from a table.
- You need to specify a condition in order to determine which specific record to delete.
- Syntax:

```
DELETE
FROM <Table Name>
WHERE <Criteria condition>
```

- Note that the field names must be in the exact same order as they appear on the table.
- Example:

```
DELETE
FROM Transaction
WHERE Transaction_ID = 6
```

Query Results:
- Deletes the record from the Transaction table whose Transaction_ID is 6.

- After table is modify, we can use a query to see if the value was committed:

```
SELECT *
FROM TRANSACTION
```

Query Results:
- A table listing all values of the Transaction table. Note that the record whose ID = 6 was deleted.
3.6 Sub-queries in SELECT Statements

3.6.1 SubQueries

- SubQueries are SQL statements embedded inside other SQL.
- SubQueries can appear inside SELECT, UPDATE, INSERT, or DELETE statements.
- Usually SubQueries are found inside the WHERE clause of a SELECT statement.
- For example:

```
SELECT * FROM TRANSACTION
WHERE (SELECT Customer_ID
       FROM Customer
       WHERE LastName = 'Clinton')
```

- Note that in the WHERE clause, since we are using the comparison operator = the result of the SubQuery must be a single value.
3.7 Joining Tables or Working with Multiple Tables

3.7.1 Joining Tables
- So far all the queries we have listed in our notes, involved one table only.
- In reality, in order to get any useful data from a database we are going to have to deal with more than one table.
- There are several commands that allow you to bring together multiple tables in a SELECT statement, and these are called Join commands. Such as equi-join, inner joins, outer joins and unions queries.
- We will be briefly discussing the equi-join, the remainder I leave up to you to research and study.

3.7.2 The Equi-Joins
- The Equi-Join allows us to concatenate or merge two or more tables.
- The queries are similar to the queries we have been creating thus far, but instead on the FROM clause, we list more than one table.
- Keep in mind that when you are joining more than one table, the query actually creates a temporary table that is the concatenation of the two or more tables.
- The Equi-Join also requires that common fields or columns exist in each of the tables are will be joined. The common field are specified in the WHERE Clause of the SELECT Statement.
- For example:

```sql
SELECT Customer.FirstName, Customer.LastName, Transaction.Tape_ID, Transaction.RentalDate, Transaction.ReturnDate
FROM CUSTOMER, TRANSACTION
```

Query Results:
- A table with information from both the Customer table and Transaction table whose ID match on both tables:

```
<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Tape ID</th>
<th>Rental Date</th>
<th>Return Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Harris</td>
<td>101</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>John</td>
<td>Harris</td>
<td>103</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
<tr>
<td>John</td>
<td>Harris</td>
<td>104</td>
<td>01/01/2001</td>
<td>01/03/2001</td>
</tr>
<tr>
<td>Gloria</td>
<td>Flores</td>
<td>105</td>
<td>01/02/2001</td>
<td>01/04/2001</td>
</tr>
<tr>
<td>Charles</td>
<td>Smith</td>
<td>102</td>
<td>01/01/2001</td>
<td>01/02/2001</td>
</tr>
</tbody>
</table>
```
3.7.3 The INNER-Join

- The Inner-Join is the most common type of join. It is similar to the equi-join but does not require the WHERE CLAUSE to indicate the criteria.
- When using an INNER-JOIN, records from both tables are used in the results ONLY if their related columns match. The related columns are specified in the Join Statement within the FROM clause or the SELECT statement.
- Syntax and example:

```
SELECT Customer.FirstName, Customer.LastName, Transaction.Tape_ID, Transaction.RentalDate, Transaction.ReturnDate
FROM CUSTOMER INNER JOIN TRANSACTION ON Customer.Customer_ID = Transaction.Customer_ID
```

- Note how the related columns are assigned in the FROM clause. This leaves the WHERE clause free for other criteria requirements.
- For example:

```
SELECT Customer.FirstName, Customer.LastName, Transaction.Tape_ID, Transaction.RentalDate, Transaction.ReturnDate
FROM CUSTOMER INNER JOIN TRANSACTION ON Customer.Customer_ID = Transaction.Customer_ID
WHERE Customer.LastName = 'Rodriguez'
```
4.1 Database Design

4.1.1 Designing a Database

- Database design is a creative process that requires a thorough understanding of database theory as well as practice and patience.
- In this section, I will cover some of the basics of database design. Since this topic is complex, only an overview will be given.
- It is assumed that by now you have taken a database course where the basic design principles were taught.

4.1.2 Normalization of a Database

Normalization Overview

- Normalization is an important topic in database design.
- Normalization consists of a set of rules that help prevent design flaws that cause repeated and inconsistent data.
- Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating two factors: redundancy and inconsistent data.
- There are three basics steps to normalization:

  - **First Normal Form:** No repeating groups
  - **Second Normal Form:** All fields or columns must depend on primary key
  - **Third Normal Form:** No non-key fields should depend on other non-key fields

4.1.3 First Normal Form

- The Rule for the **First Normal Form:** No repeating group of columns with related data
- The first step is to break the repeating groups out into a new table
- Keep the following rules in mind:

  1. Eliminate repeating groups in a table.
  2. Create a separate table for each set of related data.
  3. In the separate table, identify each set of related data with a **primary key**.

- For example consider the following design involving an auto repair shop which needs a database design. The requirements are that when clients bring in one of their vehicles, the shop registers the car and maintains a listing of the latest service date for the applicable vehicle. In database design you usually begin with one table with all the information you have acquired from interviewing the client and gathering all information necessary on the business. Suppose you create the database table un-normalized as follows:
First Normal Form Analysis:
- There are two sets of related data in this table, Client information and Vehicle information.
- By breaking up this table into two tables, the Clients table & Vehicles table.
- We establish a one-to-many relationship since a client can have multiple vehicles.
- The tables and relationships should look as follows:

![Image of relationships diagram]

### 4.1.4 Second Normal Form
- The Rule for the Second Normal Form: All fields or columns must depend on primary key.
- Records should depend on a table's primary key.
- In our example so far we have satisfied the First Normal form, but there are still a few flaws:
  - The Vehicles table has a field that does not depend on the primary key `Vehicle_Vin`, that is the `Client_ID`. A client is not related to a vehicle.
  - In addition the `Vehicle_LastServiceDate`, does not relate to the primary key as well. It relates to the visit a vehicle will make to the shop but not to the vehicle itself.
  - There is also another issue with the `Vehicle_LastServiceDate`, for each visit of a client, the user must over-write the existing last service date or create a new record, if you created a new record, you now have repeating information.
  - This design does not accommodate a log of all visits made by the client or the vehicle.

- Keep the following rules in mind:
  1. Create separate tables for sets of columns that do not depend on the primary key.
  2. Add a primary key to the newly created table.
  3. Relate the original table with the new table by a foreign key.
Second Normal Form Analysis:

- First, since the design does not accommodate a log of all visits made by the client or the vehicle, we will change the `Vehicle_LastServiceDate` field to `Visit_ServiceDate`.

Now we have two fields that do not depend on the primary key, the `Client_ID` and the `Visit_ServiceDate` fields.

- The `Visit_ServiceDate` field merely records the visit by a client with a vehicle and does not depend upon the vehicle's ID number. The `Client_ID` field is not dependent on the vehicle's ID as well.
- Therefore we need to remove these two fields from the table and create a new table named `Visit` that will keep track of all visits made by the client. Therefore each visit will be held a separate record and given a unique ID `Visit_ID`.
- We need to create relationships between these tables base on foreign keys, where the `Client_ID` of the client table will be a foreign key to the `Visit` table and the `Vehicle_VIN` will be a foreign key to the `Visit` table as well.
- Now the shop can track all of the previous visits of any particular auto or customer.
- The tables and relationships should look as follows:
4.1.5 Third Normal Form

- The Rule for the Third Normal Form: *No non-key fields should depend on other non-key fields:*
- Columns should not depend on anything other than a table's primary key. If a non-key column depends on another non-key column we need to apply the Third Normal Form.
- Our that in our example there are still a few flaws:
  - Note that Client_Discount_Description and Client_Discount_Amount both depend on Discount_Code.
  - This is a violation of the Third Normal Form, which states that non-key fields should not depend on non-key fields.

- Keep the following rules in mind:
  1. Eliminate fields that do not depend on the key by creating a separate table.
  2. Create a primary key for the new table
  3. Link the old table and new table by a foreign key

- Let's apply the Third Normal Form to the current example:

  **Third Normal Form Analysis:**
  - We remove the two values dependent on the discount code Client_Discount_Description and Client_Discount_Amount and place them on a table named *Discount.*
  - Now we create a relationship between the client table and the Discount table. This will be a one-to-many relationship where the primary key of the Discount table, will be a foreign key in the Client table.
  - The modification looks as follows:

4.1.6 Final Word on Normalization

- Normalization is very important to database design, and once you get the hang of it you can design robust databases.
- But, normalization taken too far can lead to problems
- Since normalizing increases the number of tables, this eventually can make the queries more complex due to having to search more tables thus affecting performance.
- Nothing is gained without a price!
4.2 Five Steps to Building a Database

4.2.1 Design Steps

- The following are the five steps that you should take each and every time you design a database from scratch:
  Normalization consists of a set of rules that help prevent design flaws that cause repeated and inconsistent data.

Step 1: Brainstorm
- Generate a list of every possible field that you may need to be included in your database.
- Select the data types for each field
- This step is like creating a big table listing all your fields
- You may want to create a spreadsheet with the fields and their respective data types

Step 2: Group the Fields into Tables
- Now you group the fields into logical smaller tables.

Step 3: Build Relationships Between the tables
- In this phase you build the relationships between the tables.

Step 4: Normalize the Database
- Normalize by applying the rules of normalization.

Step 5: Test your Database with Sample Data
- Add data to the tables and perform various test to make sure that referential integrity and redundancy are maintained.
Homework Assignment # 1

1. Create a Database in Microsoft Access for your Project. The database should have the following requirements:
   1) Create the necessary tables to be able to store the records for the Employee, Students, and Course Objects of your project as well as the data required for processing all functionality of the project.
   2) Apply the five design steps as best you can.
   3) With MS Access, create and apply the following queries to the database tables:

      1. List all records for Employees
      2. List all records for Students
      3. List all records for Courses
      4. Search Employee by ID
      5. Search on Student by ID
      6. Search on Course by ID
      7. Search on Student by name
      8. Search on all Courses that are available or not full >5 students
      9. Add a record or transaction to the database of a course being taken or registration
     10. Modify or add a record showing that a course was paid by the student
     11. Search for a record of a Customer rental transaction by ID
     12. List a transaction history
     13. Search for a student registration history
     14. Delete a student from the database
     15. Delete a course from the database
     16. Modify a student’s last name

4) I am leaving the details so that you can be creative on your design
5) You do not need to create forms in access or Visual Basic simply creates the queries necessary to carry out the operations.
6) You will bring the database file with queries etc. to class and demonstrate it to the professor.