8. DATABASES

8.1 WIKIPEDIA ON DATABASES:

A database management system (DBMS) is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. A DBMS includes:

- A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.
  The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used. A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).
  The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.
- Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).
- A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data. It also controls the security of the database.
- Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.

If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance). It also maintains the integrity of the data in the database.

The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database.

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.

DBMS's are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

One can characterize a DBMS as an "attribute management system" where attributes are small chunks of information that describe something. For example, "color" is an attribute of a car. The value of the attribute may be a color such as "red", "blue", "silver", etc. Lately databases have been modified to accept large or unstructured (pre-digested or pre-categorized) information as well, such as images and text documents. However, the main focus is still on descriptive attributes.

DBMS roll together frequently-needed services or features of attribute management. This allows one to get powerful functionality "out of the box" rather than program each from scratch or add and integrate them incrementally. Such features include:

Querying is the process of requesting attribute information from various perspectives and combinations of factors. A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data. It also controls the security of the database. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets.

When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency. In the relational model, related records are linked together with a "key".

For instance, a common use of a database system is to track information about users, their name, login information, various addresses and phone numbers. In the navigational approach all of these data would be placed in a single record, and unused items would simply not be placed in the database. In the relational approach, the data would be normalized into a user table, an address table and a phone number table (for instance). Records would be created in these optional tables only if the address or phone numbers were actually provided.

Linking the information back together is the key to this system. In the relational model, some bit of information was used as a "key", uniquely defining a particular record. When information was being collected about a user, information stored in the optional (or related) tables would be found

by searching for this key. For instance, if the login name of a user is unique, addresses and phone numbers for that user would be recorded with the login name as its key. This "re-linking" of related data back into a single collection is something that traditional computer languages are not designed for.

The relational approach requires loops to collect information about any one record. Codd's solution to the necessary looping was a set-oriented language, a suggestion that would later spawn the ubiquitous SQL. Using a branch of mathematics known as tuple calculus, he demonstrated that such a system could support all the operations of normal databases (inserting, updating etc.) as well as providing a simple system for finding and returning sets of data in a single operation.

8.2 FINANCIAL DATA

Financial analysis requires data. Furthermore, when doing quantitative financial analysis, we often find that the data, such as historical price or fundamental data, comes in a very simple database structure such as a flat-files or worse, a spreadsheet. Flat files are a primitive database design and often contain from redundant and inconsistent data. Those who understand the elements of good database design certainly avoid the flat file structure for all but the simplest data, like historical price data.

Financial analysts are almost always experienced in Excel. And they should be. Excel is the most rapid development environment for building and testing financial models. But, Excel is not a relational database and it should not be used as such. Data in Excel is easily corrupted, and too much data in a spreadsheet has been known to overflow memory and cause crashes. Furthermore, production systems run in Excel almost always suffer from quality problems. If you are collecting and/or analyzing financial markets data, we suggest you use the right tool for the job—a relational database.

It is difficult, however, to break away from Mother Excel. Using Excel, it's easy to paint a range of returns for example and pass it to the covariance function. Being able to see it the data and the function calls is very comforting. If we use a database, however, we cannot see it and this can be somewhat scary. But, fear not. In the long run, we are much better off. There are far more advantages to using a database than disadvantages. Using databases and .NET programs gives us much more control and security when building production systems.

Visual Basic.NET is an efficient platform for creating financial applications that interact with data and databases. The front-end applications insulate the user from the back-end complexities and inner workings of the relational database management system and protect the data from accidental alteration or deletion.

Just remember, in very general terms, a database stores information and provides methods for managing the data, including methods to retrieve existing data, add new data and edit data. As in the case of a flat file, a database may be a very simple construct, consisting of a single file. But, as we grow in our understanding of databases, we will see that they can become very powerful, full-scale client/server relational database management systems. So, it is important that we learn how relational databases work in order to accomplish more advanced analysis. As with other topics discussed in this book, we cannot hope to cover all the topics relating to relational databases. There are hundreds of books on the market that deal with this topic alone. But, we will be able to discuss several relatively advanced topics and build three or four models that demonstrate relational database design and connectivity as used in quantitative research.

Let's consider a financial markets example using a relational database in the front office. On a trading desk, we may want to attribute trading profits and losses to different factors so as to assess the success of an automated trading system. But, profit and loss analysis and risk management requires combining data in a relational way with historical trade and price data along

with profit and loss information. This scenario requires the use of a relational database management system.

In general there are two types of databases used in financial markets: operational databases and analytical databases. Operational databases store dynamic data such as portfolio and position information. Analytical databases hold static data such as historical price or trade history data often in a flat file. Regardless of the brand of database software, both of these types of databases will be managed using the relational database model (RDM). (Hernandez, p. 3)

In the RDM, data is held in tables, which are made up of columns, called "fields," and rows, called "records." Connections between different tables are defined in relationships. These relationships between tables are established through shared fields. The RDM has the advantage that, through its use of tables and relationships between them, data integrity is ensured and consistency and accuracy of data is guaranteed. Furthermore, changes in the design of the database will not adversely affect the .NET applications that we build to access it. (Hernandez, p. 16)

Programs that we create in .NET can interact with databases through a set of objects known as MS ActiveX Data Objects (ADO) and a specialized language called Structured Query Language (SQL). SQL enables us to talk to a database from a .NET application. SQL is the industry wide standard for interacting with databases for everything from simple data retrieval, called queries, to modification and updating of data, and even database creation. Regardless of which relational database management system you are using, SQL will be the language you will use to carry on a conversation in code. Relational database management systems (RDBMS) are software applications for building, managing and modifying relational databases. The most popular large scale RDBMSs are from Microsoft, Oracle and Sybase. This may be a lot of new information, if you are not familiar with databases, but don't worry, in Chapters 12 and 13, we will look at ADO and SQL in greater depth.

In order to fully understand the RDM and the following chapters in this book, it is imperative that become familiar with "database speak," the terms and phrases used in the database industry. We have used some of the terms already—tables, columns, rows, relationships. Over the next few pages, we will define and briefly discuss some of the more important terms. Afterwards, we will look at the three databases included on the CD that will illustrate most of these terms.

8.3 TABLES

A table is the primary structure in a relational database. It consists of columns and rows, often called fields and records. Tables represent things, like historical data for IBM, and events like trades. Tables should then have names, which describe the data they hold, like IBMData, or just IBM, or OptionTrades. Further, tables can either be data tables, which supply information such as historical prices, or validation tables, which implement data integrity. For example, we may have a table that contains a list of options expiration dates.

8.4 FIELDS

A field, or column, represents a characteristic of a record. For example, our IBMData table would probably have a ClosePrice field. Fields then have names, data types, and lengths. Data in databases can be alphanumeric, numeric, or date/time. Also, fields can contain distinct or multipart values and may have values that are calculated.

8.5 RECORDS

A record, or row, holds the actual data in a table. A single record in a table is made up of one row containing all of the columns in the table including a primary key that uniquely identifies a

record. So, January 21, 2003 identifies a unique record, or data point, of IBMData. The full record contains the date, open, high, low, and closing prices and the volume.

8.6 PRIMARY KEYS

Primary keys are special fields that uniquely identify a record in a table. So, as in the previous example, the Date field represents a unique record in a table of historical prices. Every table in a database must have a primary key and no two tables should have the same primary key. Primary keys ensure that each record in a table is uniquely identifiable. Therefore, each element of the primary key field must be unique and cannot be null. So, no duplicate dates would be aloud in our example.

8.7 FOREIGN KEYS

Foreign keys establish relationships between pairs of tables. Relationships between two tables arise when the primary key column in one table is identical to the foreign key column in the other. In a later example, we will see a graphic depiction of a relational database showing the primary and foreign keys for different tables along with arrows representing the relationships between them.

8.8 RELATIONSHIPS

As we have seen, relationships are connections between pairs of tables, through the use of primary and foreign keys. There are three different types of relationships: one-to-one, one-to-many, and many-to-many.

8.9 ONE-TO-ONE RELATIONSHIPS

A relationship is said to be one-to-one if a single record in the first table is related to a single record in the second table, and vice versa.

8.10 ONE-TO-MANY RELATIONSHIPS

A relationship is said to be one-to-many if a single record in the first table can be related to several records in the second table, but at the same time a single record in the second table can only be related to only a single record in the first table. It may seem a little confusing right now, but a later example will make this idea quite clear.

8.11 MANY-TO-MANY RELATIONSHIPS

A relationship is said to be many-to-many if a single record in the first table is related to many records in the second table, and vice versa. In the case of a many-to-many relationship, we need to create a linking table by copying the primary key from each table into the new table. We suggest you find a good book on database design if you attempting to build complex databases that include many-to-many relationships.

8.12 QUERIES

A query is an SQL statement used to retrieve rows of information from one or more tables. Queries will often also contain search criteria to limit the amount of data returned from the tables. For example, we may create a query that retrieves the trade data only for the month of March, 2001.

SQL queries also allow us to join tables. Joining tables enables use of data from several tables in a relational database for a single purpose. For example, we could display a single column from one table or several columns from multiple tables in a single query. From this, you may begin to see the flexibility and power of relational databases.

8.13 NORMALIZATION

Often you will hear database professionals talk about normalization or normal forms. Normalization is the process of breaking down a large table or tables into smaller tables in order to eliminate duplication of data and to prevent certain problems that commonly arise with database interaction. A normal form is a set of rule s that test a table structure to ensure it is sound and free of errors. There are at least five normal forms--first through fifth—used to test for specific sets of problems. Tables we will use are in at least third normal for since each one has a primary key that uniquely identifies each record.

8.14 DATABASE DESIGN

Creating proprietary databases from scratch is no small task. It necessitates examinations of the business purposes of the database as well as the technical means to implement them. In short, designing relational databases requires a process or methodology. Doing so without one can lead to disaster. Again, several good books on relational database design have already been written, so we will quickly review the process. Michael Hernandez in his book Database Design for Mere Mortals outlines a seven phase process for database design.

- Define the purpose of the database and the tasks that users will perform against it.
- Analyze current database solutions.
- Create tables, fields and primary keys that characterize the subjects the database will track.
- Determine the relationships that exist between tables.
- Define the constraints or business rules for the data.
- Develop ways to look at or view the data.
- Review the integrity of the data, including checking the field specifications, testing the validity of relationships, and reviewing the business rules.

Well-designed databases are easy to modify structurally, allow for efficient retrieval of data and make it easy for developers to build applications to connect to it. (Hernandez, p. 28)

Access Database

MS Access databases are relational databases supported by all Microsoft Windows environments. You do not need to have MS Access software installed on your computer to interface with Access databases through .NET. In an Access database, all the various parts of the database are stored in a single file, which has an .mdb extension. The CD contains three Access databases—Finance.mdb, DirtyFinance.mdb, and Options.mdb—that we will use over the course of the remainder of the book. If you have MS Access software on your computer, feel free to open these database in Access and examine their structures. Let's take a look at each of them.

8.15 THE FINANCE.MDB DATABASE

Finance.mdb is an MS Access database included on the CD with this book that uses flat files to hold daily historical price data for 13 stocks and the S&P 500. The individual data tables in Finance.mdb are named AXP, GE, GM, IBM, INTC, JNJ, KO, MCD, MO, MRK, MSFT, SUNW, WMT and SPX. In addition, there is a validation table named Tickers, which contains the 13 stock ticker symbols shown.

The 14 data tables consist of the primary key column, the Date, and five other columns named OpenPrice, HighPrice, LowPrice, ClosePrice and Volume. Each table holds 12 years of daily price data from January 2, 1990 to December 31, 2002. Here is a sample of the IBM table showing the structure:

Date	OpenPrice	HighPrice	LowPrice	ClosePrice	Volume
2-Jan-90	23.54	24.38	23.48	24.35	1760600
3-Jan-90	24.53	24.72	24.44	24.56	2369400

4-Jan-90	24.6	24.94	24.56	24.84	2423600
5-Jan-90	24.81	25.25	24.72	24.78	1893900
8-Jan-90	24.66	25.06	24.66	24.94	1159800
2-Jan-90	23.54	24.38	23.48	24.35	1760600

The Tickers validation table consists of a single column named Symbols, which holds the ticker symbols for each of the 13 stocks. Here is a sample of the Tickers table:

Symbols
AXP
GE
GM
IBM
etc.

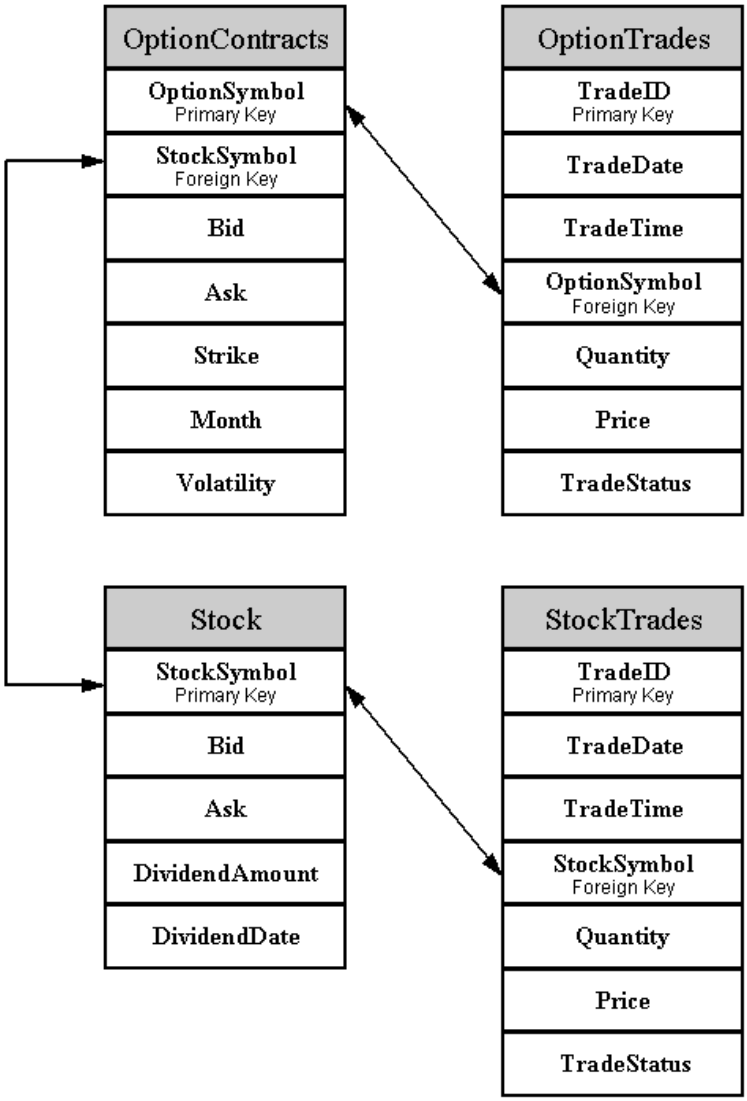
We have made every attempt to ensure that the data in the Finance.mdb database is clean and free from errors. This is not the case with the DirtyFinance.mdb database.

8.16 THE DIRTYFINANCE.MDB DATABASE

The DirtyFinance.mdb Access database included on the CD purposely contains dirty data. It is identical in every way structurally to the Finance.mdb data. The only difference is that we have gone through and corrupted the data using all kinds of sly and malicious techniques. But, the errors we have created are typical of those you will encounter in real data purchased from data vendors. In Chapter 14 it will be your job to build a .NET program that finds the dirty data and cleans it.

8.17 THE OPTIONS.MDB DATABASE

The Option.mdb Access database uses a relational database structure to hold information about stocks and options as well as stock trades and option trades. In fact, there are four tables in the Options.mdb database representing each of these things—Stocks, OptionContracts, StockTrades and OptionTrades. As we saw earlier, the relationships between two tables in a relational database are made possible by common primary and foreign keys. In Options.mdb, for example, the Stock and StockTrades tables are related through a StockSymbol primary key in the Stock table and the foreign key StockSymbol column in the StockTrades table. Here is a diagram showing the structure or schema of the Option.mdb database. In this diagram, the relationships are represented by arrows.



All of the relationships in the Options.mdb database are one-to-many. As you may be able to gather from the diagram, a one-to-many relationship exists between the Stock and OptionContracts tables. Clearly, a single stock can have many options contracts on it. But in the opposite direction, it is not the same. A single option contract can have only one underlying stock associated with it.

Earlier in the chapter, we briefly described a many-to-many relationship between two tables. Although not represented in the Options.mdb diagram, let's consider a quick example. A single option contract may be involved in many trades, but an individual trade could have more than one option contract associated with it if we assume spreads are included in a SpreadTrades table. In this way, a single OptionContract could be related to several SpreadTrades, and a single SpreadTrade could be related to several OptionContracts.

8.18 SUMMARY

When doing financial modeling and certainly when building production trading and risk management systems, relational databases are superior to Excel as a way to store and manage data. The database field has its own language that we must learn before we can begin creating

and interact with them. In this chapter, we looked at and defined several database terms. Furthermore, creating new relational databases necessitates the use of a design methodology. We very briefly reviewed the seven steps of a well-known methodology.

There are three Access databases included on the CD with this book—Finance.mdb, DirtyFinance.mdb and Options.mdb. We will be building .NET windows applications in later chapters that access them.

CHAPTER PROBLEMS

- What are operational and analytical databases?
- What is SQL?
- Describe tables, rows and columns.
- What are relationships and how are they created? Describe the three types of relationships.
- What is the process to go through to design a relational database?

PROJECT ONE

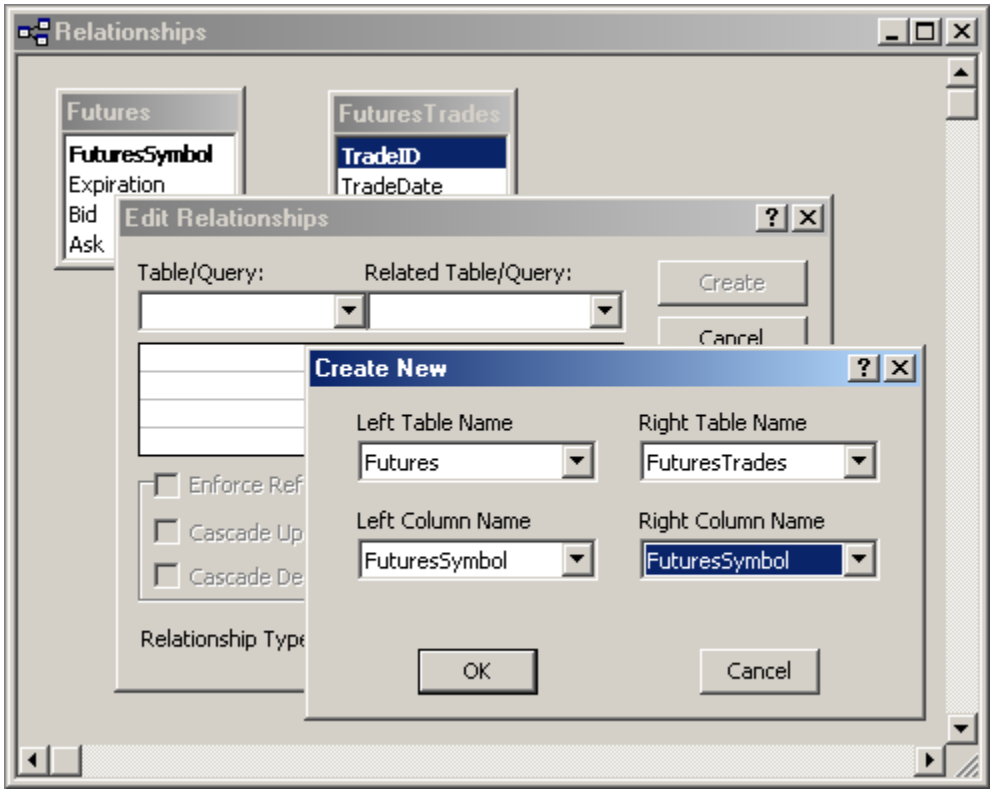
Assuming you have it, create a simple relational database called Futures.mdb in MS Access. This database should consist of two tables named Futures and FuturesTrades. The Futures table should have columns named FuturesSymbol, Expiration, Bid and Ask. The FuturesTrades table should have columns named TradeID, TradeDate, TradeTime, FuturesSymbol, Quantity and Price.

In Access, open a blank Access database. Next, under Objects click on Tables, and then on New. In Design View, enter the column names for the Futures table. On the FuturesSymbol field, right click and select Primary Key. Close the Design View window and name this table Futures.

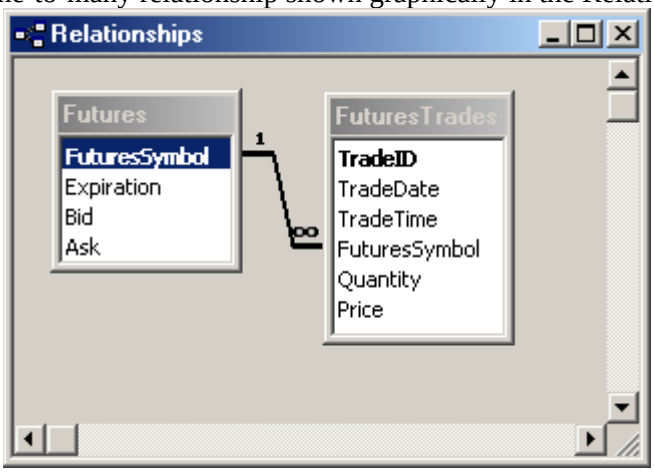
Next, click on New again. In Design View, enter the column names for the FuturesTrades table. Set TradeID as the primary key. Close the Design View window and name this table FuturesTrades.

Under the Tools menu bar item, select Relationships. Add both the Futures and FuturesTrades tables.

On the menu bar, select Relationships and the Edit Relationships. In the Edit Relationships window, click on Create New. Add a relationship between FuturesSymbol field in the Futures table and the FuturesSymbol field in the FuturesTrades table as shown in the picture.



Back in the Edit Relationships window, click on Enforce Referential Integrity and Create. You should now see the one-to-many relationship shown graphically in the Relationships window.



Now, try adding some hypothetical data to the tables by Openning the table.

PROJECT TWO

Design a relational database to hold bond trading data and create it in MS Access. Your database should contain at least two tables related to one another in a one-to-many way.