

Module 1

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 1 Study Guide and Deliverables

- | | |
|------------------------|--|
| Background Concepts | <ul style="list-style-type: none">• Coronel & Morris, chapters 1 and 2 |
| Readings: | |
| Optional SQL Readings: | <ul style="list-style-type: none">• <i>12th Edition</i>: Coronel & Morris, sections 7.1 through 7.4 of chapter 7• <i>13th Edition</i>: Coronel & Morris, sections 7.1 through 7.3 of chapter 7, sections 8.1 and 8.2 in chapter 8 |
| Assignments: | <ul style="list-style-type: none">• Term Project Iteration 1 due Tuesday, September 13 at 6:00 AM ET• Lab 1 due Tuesday, September 13 at 6:00 AM ET |
| Live Classroom: | <ul style="list-style-type: none">• Tuesday, September 6 from 8:00-9:30 PM ET• Wednesday, September 7 from 8:00-9:30 PM ET |

Which First?

Read the book chapters before reading the online lectures.

Terminology Note

The more common name for what our text terms the *internal model* is the *logical model*.

Section 2.4.3 on page 37 of the textbook instructs you to prefix each attribute name with the name of its containing entity. However, prefixing attribute names in this manner was necessary decades ago when many DBMS would not support having the same column name in multiple tables. A better contemporary practice is to name columns in different tables the same when the columns represent the same attributes of different types of objects. In these cases the domains and data types should be identical. Best practice is to explicitly name the table for each column in a request—for example

```
SELECT My_table.my-column.
```

Introduction

metcis669_09_sp1_bshdy_lec01 video cannot be displayed here

Let's begin by defining some database systems terminology.

Data are raw facts, meaning facts have not yet been processed to reveal their meaning.

Data is processed to yield *information* on which business decisions can be based.

Data management is a discipline that focuses on the proper generation, storage, processing and retrieval of data. Typically, data management requires the use of a computer database.

A *database* is a shared, integrated computer structure that houses a collection of end-user data, and metadata about the stored data. A well-designed database facilitates data management and becomes a valuable information generator. A poorly designed database is likely to lead to errors in processing data and to bad decisions.

Let's summarize some key points:

- Data constitute the building blocks of information.
- Information is produced by processing data.
- Information reveals the meaning of data.
- Good, relevant, and timely information is the key to good decision making.
- Good decision making is critical to organizational survival in a competitive environment.

What a Database Is

A database is a shared, integrated computer structure that houses a collection of user data and metadata. *Metadata* is data about data, such as descriptions of the relationships that exist among data. A *database management system* (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database. Databases are classified according to the number of users, the database locations, and the expected type and extent of use. The most popular way to classify databases is by the use and timeliness of the data. For example, a production database contains up-to-the-minute real-world information while a warehouse database stores data for making decisions.

Basic Terminology	
data	Raw facts from which the required information is derived. Data are not very useful unless they are organized in a logical manner.
field	Data that represents a specific characteristic of an entity. A field may contain a string of characters, a number, a date, or another data type. A field may represent a telephone number, name, date, weight, or other specific characteristic that the end user or application wants to keep track of.
record	A logically connected set of one or more fields that describes an entity such as a person, place, event, or thing. For example, a CUSTOMER record may be composed of the fields customer_number, last_name, first_name, city, state, post_code, and phone. There is one record for each individual instance of an entity type. For example, there is one record for each customer.
file	Historically, a <i>file</i> was a collection of file folders, properly tagged and kept in a filing cabinet. Although such manual files still exist, we more commonly think of a (computer) file as a sequence of bytes or related records that contain information of interest. For example, a sales organization is likely to keep a file containing customer data. Keep in mind that the phrase “related records” reflects a relationship based on function. For example, customer data are kept in a file named Customer. The records in this Customer file are related by the fact that they all pertain to customers. Similarly, a file named Product would contain records that describe products: the records in this file are all related by the fact that they all pertain to products. You would not expect to find customer data in a product file, or vice versa.

Table 1.4 Sample Customer File Fields

FIELD	CONTENTS	SAMPLE ENTRY
CUS_LNAME	Customer last name	Ramas
CUS_FNAME	Customer first name	Alfred
CUS_INITIAL	Customer initial	A
CUS-AREACODE	Customer area ode	615
CUS_PHONE	Customer phone	234-5678
CUS_ADDRESS	Customer street address or box number	123 Green Meadow Lane
CUS_CITY	Customer city	Murrfreesboro

CUS_STATE	Customer state	TN
CUS_ZIP	Customer zip code	37130

Different types of databases that you may find in the workplace can be classified as follows:

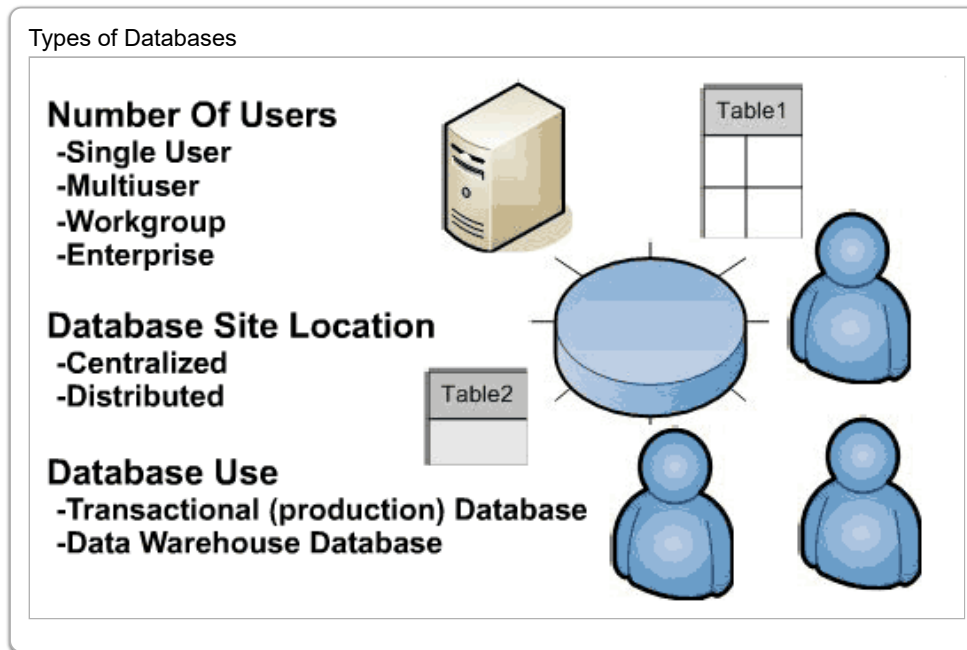
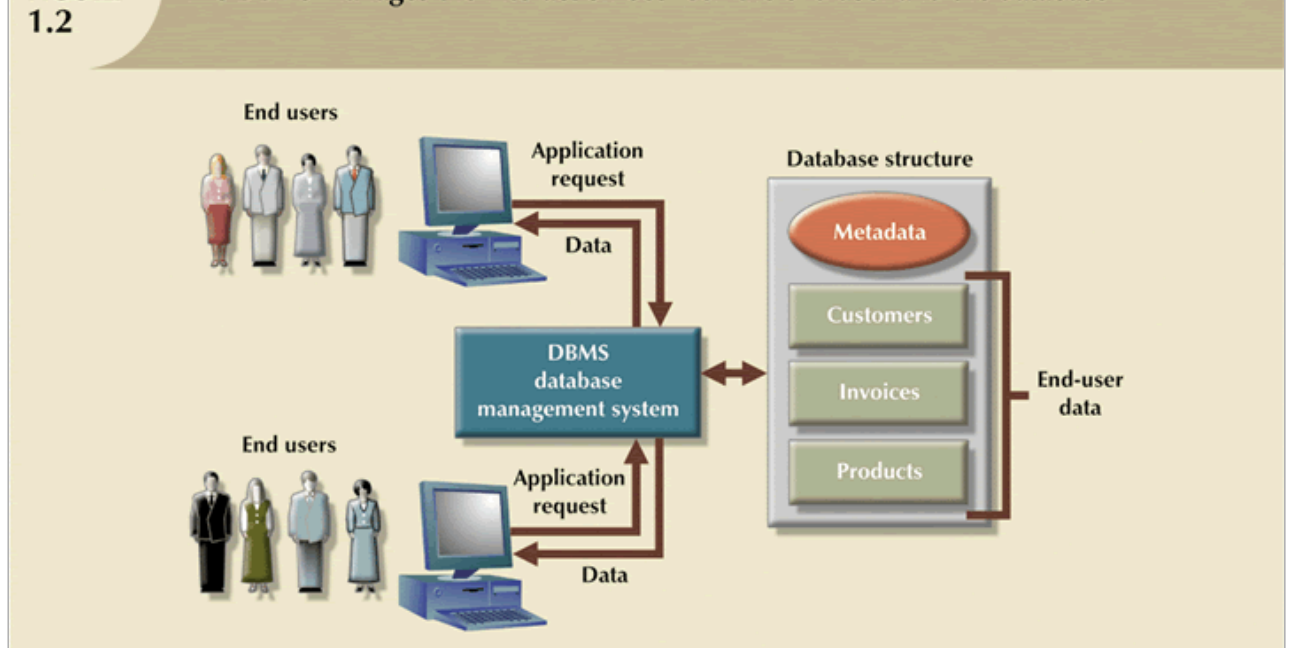


FIGURE 1.2

The DBMS manages the interaction between the end user and the database



Test Yourself 1.1

Which of the following are true of a database? (Check all that apply.)

A *database* is a shared, integrated computer structure that houses a collection of end-user data and metadata about the stored data.

This is true. A database is a collection of data for end-users to access concurrently and easily.

A *data warehouse* is primarily concerned with processing business transactions.

This is false. A warehouse database stores data for decision-making. A production database is involved in processing business transactions.

A *production database* is typically transactional, meaning there are numerous updates occurring so the database is up-to-the-minute.

This is true. Transactional databases are used primarily to support day-to-day operations and decision making, while data warehouses are used primarily for making decisions, especially based upon long-term data.

Databases can be classified by the number of users, location, and use.

This is true. Databases are classified through the number of users, site location, and database use. Example database categories include decision support, transaction processing, enterprise, workgroup, and single-user.

A database structure consists of metadata and end-user data.

This is true.

Why Database Design is Important

Imagine what would happen if someone started building a house without any kind of blueprint or design. They might throw up some walls here, a window or door there, with some plumbing and electrical wires thrown in, and end up with a mess in the end! In the same way, a database created without a design will end up as a mess. Just like a blueprint is created for a house before it is built, a database is designed before it is created.

Additionally, there are good database designs and bad database designs. Good database designs eliminate or mitigate data anomalies by avoiding data redundancy. Redundant data is oftentimes the source of difficult-to-trace errors. For example, a customer's telephone number may be found in the customer file, in the sales agent file, and in the invoice file. Different reports might give different results for those same pieces of data. Good database designs are also *scalable*, that is, the database performs well as its load (sometimes significantly) increases. The total volume of data stored, the number of database users, the rate at which data comes in, the number of different types of data, are all factors that can increase over time, and a scalable design will handle all of these well.

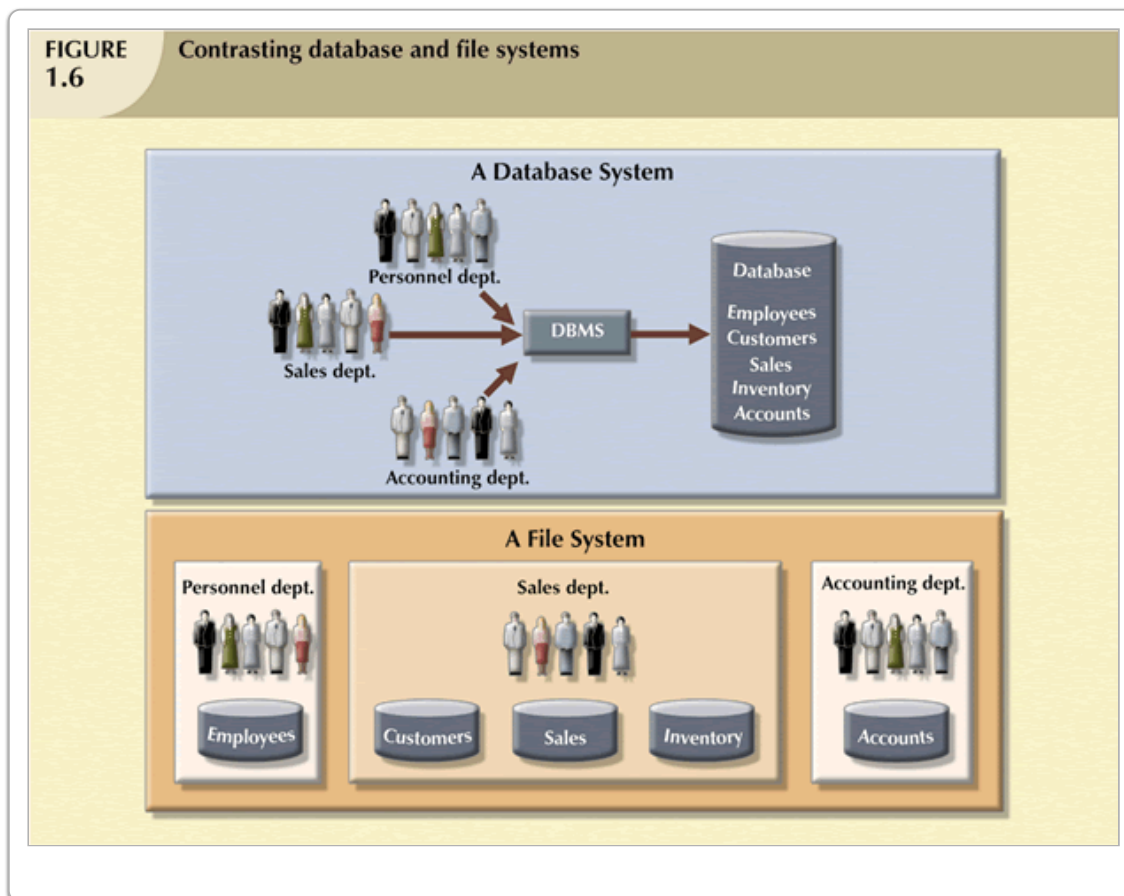
To summarize:

- Well designed databases are critical to the correct functioning and successful management of enterprises.
- Applications cannot easily overcome the problems of bad database designs.
- The existence of a DBMS does not guarantee good data management, nor does it ensure that the database will be able to generate correct and timely information.

- Good database designs are scalable.
- Ultimately, the end user and the designer decide what data will be stored in the database, and how that data is to be organized in the database.

How Modern Databases Evolved

File systems were widely used before the development of database software systems. Historically, the first computer applications focused on clerical tasks such as order entry processing, payroll, and work scheduling. Such applications accessed data stored in files. Database models were developed to address the inherent weaknesses in these file systems. Rather than deposit data into different files, databases keep data within a single repository, enabling tighter control to be maintained over data-related activities.



Flaws in File System Data Management

File systems lack key data management features that today's DBMS software provides. This makes data management with file systems difficult and cumbersome. Each file typically required its own set of data management programs. Many files would suffer from data redundancy leading to inconsistencies, anomalies, and lack of data integrity. Each file was used by many application programs. This meant that a mature file-based data system might require hundreds or thousands of programs.

Serious problems resulted from this data dependency. Access to any file was dependent on data characteristics and storage formats. Even a minor change to a data structure within a file would require changing all programs accessing that data.

How Database Systems Differ from File Systems

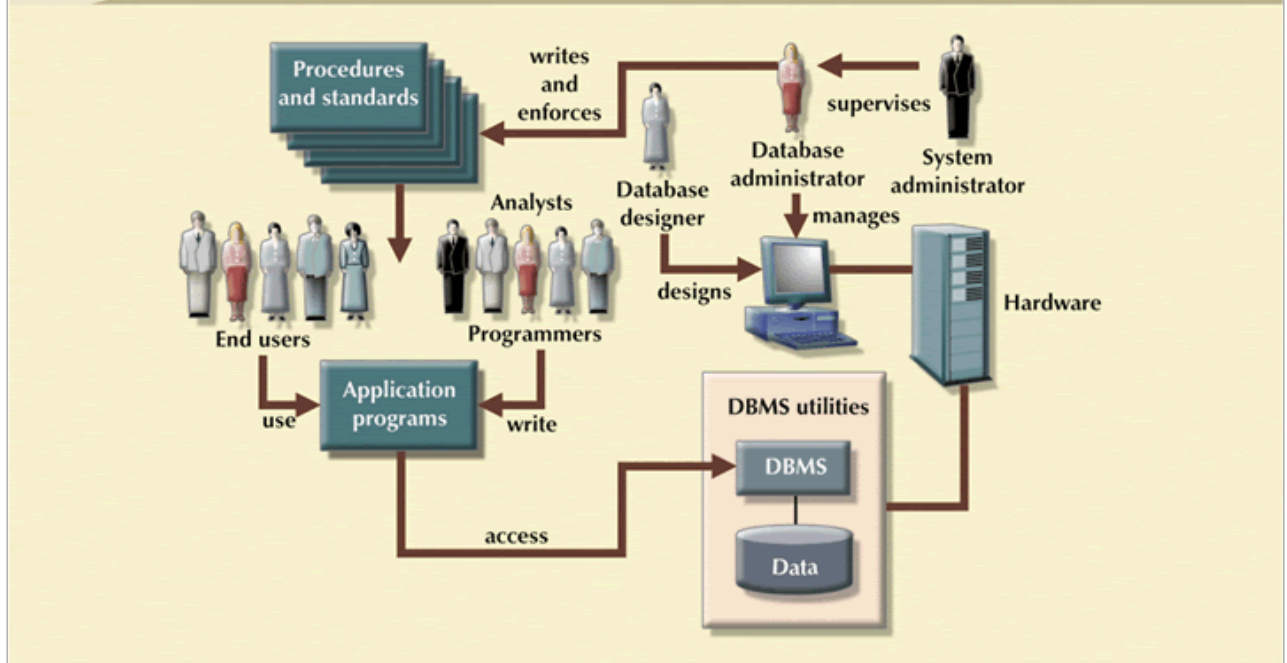
A database management system (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database. The DBMS provides a query language such as the Structured Query Language (SQL), which users and software applications use to access and modify the data. The DBMS creates an environment in which end users have better access to more and better-managed data. This promotes an integrated view of the organization's operations. The DBMS receives user requests and translates these requests into the computer programs required to fulfill the requests. In this way, the DBMS hides much of the database's internal complexity.

The functions of a current-generation DBMS may be summarized as follows:

- The DBMS stores the definitions of data and their relationships (metadata) in a system catalog; any changes made are automatically recorded in the system catalog.
- The DBMS creates the complex structures required for data storage.
- The DBMS transforms entered data to conform to the data structures.
- The DBMS creates a security system and enforces security within that system.
- The DBMS allows multiple users to have concurrent access to the data.
- The DBMS performs backup and data-recovery procedures to ensure data safety.
- The DBMS promotes and enforces integrity rules to eliminate data-integrity problems.
- The DBMS provides access to the data via utility programs and programming language interfaces.
- The DBMS provides access to data within a computer network environment.

The components of a Database environment are hardware, software, people, procedures, and data, as illustrated in the figure below:

FIGURE 1.7 The database system environment



Test Yourself 1.2

What are the advantages of a database system over a file system? (Check all that apply.)

A database system allows multiple users to access the data concurrently.

This is true. The main feature of a database is concurrency control, which allows multiple users to access the data concurrently.

A database system typically has its own set of data management programs for each table.

This is false. A database management system is a collection of programs that manages the database structure.

A database system keeps data in a single repository, enabling better data management.

This is true. A database has a single repository.

A database system has integrity controls to help prevent data-integrity problems.

This is true. One of the main functions of a database is to assure data integrity.

Jobs in the Database Field

A quick search of monster.com or bostonworks.com will reveal hundreds of jobs available in the database field. Let us consider the following jobs:

Database Administrator

- Focused on individual databases and DBMSs
- Need strong technical skills in specific DBMSs

Data Administrator

- Plans for databases and technology
- Sets standards for data such as privacy and risk of loss
- Works with computerized and non-computerized databases

Database Modeler/Analyst/Designer/Programmer

- Responsible for design and implementation of databases and the application systems that interface with a DBMS.
- The Modeler's primary responsibility is gathering the data requirements and representing them in the data model
- The Designer may participate in the modeling, and translates the model into an operational database, often with the assistance of system and storage administrators.
- Application analysts gather, document and coordinate the application and user requirements.
- Programmers write the software applications, based on the application and data requirements. There are several programmer specialties, including database programmers who write stored procedures, triggers, and other code that resides in the database. The SQL code associated with the application may be written by application programmers or by specialists.

Practice Questions and Answers

Given the file structure shown in Figure P1.1, answer Problems 1.3 through 1.6 as a self-paced exercise.

FIGURE P1.1: The File Structure for Problems 1.3-1.6

PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd. Gainesville, FL 37123	\$16,833,460.00
25-2D	Jane D. Grand	615-898-9909	218 Clark Blvd., Nashville, TN 36362	\$12,500,000.00
25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$32,512,420.00
25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd.	\$21,563,234.00

			Gainesville, FL 37123	
27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$10,314,54.00
29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd. Gainesville, FL 37123	\$25,559,999.00
31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	\$56,850,000.00

Test Yourself 1.3

How many records does the file contain, and how many fields are there per record?

The file contains seven records (21-5Z through 31-7P) and each of the records is composed of five fields (PROJECT_CODE through PROJECT_BID_PRICE.)

Test Yourself 1.4

What problem would you encounter if you wanted to produce a listing by city? How would you solve this problem by altering the file structure?

The city names of projects are not explicitly recorded in the file. We could infer that the cities of the projects are the same as those of the MANAGER_ADDRESS attribute; decomposing this character (string) field at the application level is cumbersome at best. (Queries become much more difficult to write and take longer to execute when internal string searches must be conducted.) If the ability to produce city listings is important, it is best to store the city name as a separate attribute.

Test Yourself 1.5

If you wanted to produce a listing of the file contents by last name, area code, city, state, or zip code, how would you alter the file structure?

The more we divide the address into its component parts, the greater its information capabilities. For example, by dividing MANAGER_ADDRESS into its component parts (MGR_STREET, MGR_CITY, MGR_STATE, and MGR_ZIP), we gain the ability to easily select records on the basis of zip codes, city names, and states. Similarly, by subdividing the MANAGER name into its components MGR_LASTNAME, MGR_FIRSTNAME, and MGR_INITIAL, we gain the ability to produce more efficient searches and listings. For example, creating a phone directory is easy when you can sort by last name, first name, and initial. Finally, separating the area code and the phone number will yield the ability to efficiently group data by area codes. Thus MGR_PHONE might be decomposed into MGR_AREA_CODE and MGR_PHONE. The more you decompose the data into their individually useful component parts, the greater the search flexibility. Data that are decomposed into their most basic components are said to be *atomic*.

Test Yourself 1.6

What data redundancies do you detect, and how could these redundancies lead to anomalies?

Note that the name, address, and phone number of the manager named Holly B. Parker occurs three times, indicating that she manages three projects coded 21-5Z, 25-9T, and 29-2D, respectively. This presents several problems. One problem is that the names, addresses, and phone numbers of Holly Parker can be different in the different records. This makes it difficult to assure that the data is consistent and correct in all places that it is stored. Another problem is that this file representation is larger than it needs to be, because the names, addresses, and telephone numbers are stored multiple times. The same problems exist for the multiple occurrences of George F. Dorts. (The occurrences indicate that there is a 1:M relationship between PROJECT and MANAGER: each project is managed by only one manager but, apparently, a manager may manage more than one project.) Ms. Parker's phone number and address also occur three times. If Ms. Parker moves and/or changes her phone number, these changes must be made more than once and they must all be made correctly...without missing a single occurrence. If any occurrence is missed during the change, the data are "different" for the same person. After some time, it may become difficult to determine what the correct data are. In addition, multiple occurrences invite misspellings and digit transpositions, thus producing the same anomalies.

Summary

In this lecture we have learned that databases are shared integrated computer systems based on database management systems, and that databases contain both end user data and metadata, which describes the data and the relationships between the data. We learned that there are different types of databases. Operational databases operate in real time processing business transactions. Decision support databases do not process real time business transactions; instead, they analyze data from operational databases in order to support decision making. We learned that database systems evolved from file systems, and that there are many differences between file systems and database systems. Database systems have additional features, including metadata that describes the relationships between the data. The ability to enforce constraints on the data, concurrent access and transactions, security and recovery features, and the ability to access the data using flexible portable query languages over computer networks.

Test Yourself 1.7

Select all that are true of databases. (Check all that apply.)

A poorly designed database could have performance problems and difficult-to-trace errors that could cause the application software to behave incorrectly.

This is true. It is important to have a good database design to assure that there won't be performance problems in the future.

The DBMS helps create an environment in which end users have better access to more and better-managed data.

This is true. DBMSs allow concurrent access between multiple users.

A file system or spreadsheets do not support database functionalities like concurrency control or data integrity.

This is true. File systems and spreadsheets do not support most DBMS functions.

■ Lecture 2 - Data Models

Introduction

metcis669_09_sp1_bshdy_lec02 video cannot be displayed here

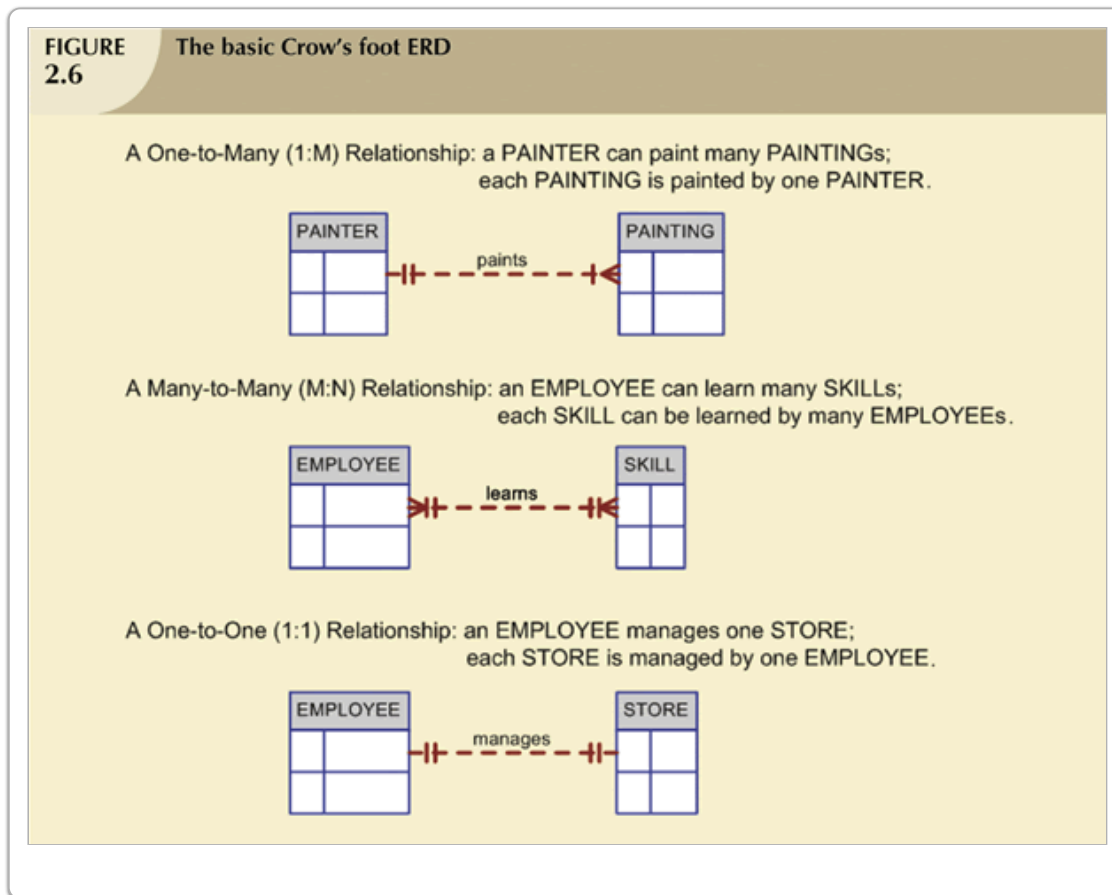
A data model is a collection of concepts that can be used to describe the structure of a database. A data model provides abstraction, permitting database designers to discuss the data independent of the physical aspects of a database management system implementation. The main function of a data model is to help us understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, and transformations. A data model provides a foundation for good database design. A data model provides a blueprint of the data that is required for a functional system. Data models capture key objects and even behaviors that can be implemented in a DBMS. Data models facilitate communication between users, database designers, and application programmers, who often have different understandings of the data and their relationships. Data models facilitate this critical communication by providing understandable yet expressive and precise ways of describing the fundamental data objects and their relationships, and well defined ways to translate those abstractions into concrete detailed database designs.

There are three main categories of data models:

1. High-level or *conceptual* data models, which are based on entities (objects) and relationships.
2. Low-level or *physical* data models, which are specific to particular DBMS such as Oracle.
3. Representational or implementation data models, which are also termed *logical* data models.

The distinction between conceptual and logical data models was clear in the early days of hierarchical and network databases, where the physical implementation of the database was very different than the conceptual model. With each successive generation the ability of databases has improved to directly implement the abstractions in the conceptual models and the number of differences between conceptual and implementation data models has decreased. Today with object-oriented models implemented on object-oriented databases the conceptual and implementation models may be virtually identical.

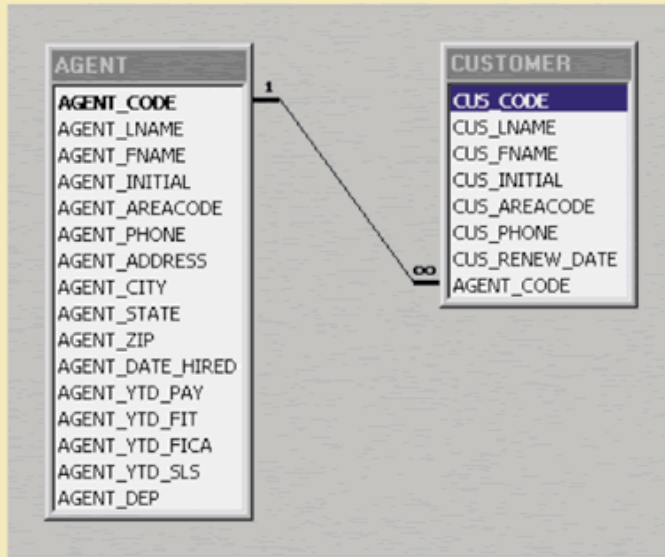
A data model is needed to capture the data requirements and validate them with the users. The diagram below illustrates a simple conceptual data model.



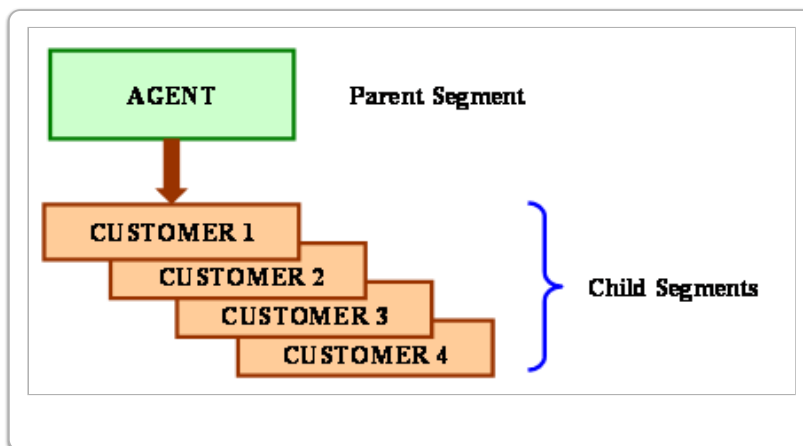
Data models are of different kinds. Let us look at relational, hierarchical and network data models. If the relationship between AGENT and CUSTOMER (one agent has many customers) was implemented in a relational model it would look like the figure below. Note that attributes don't have to be represented in the early stages of modeling.

**FIGURE
2.4**

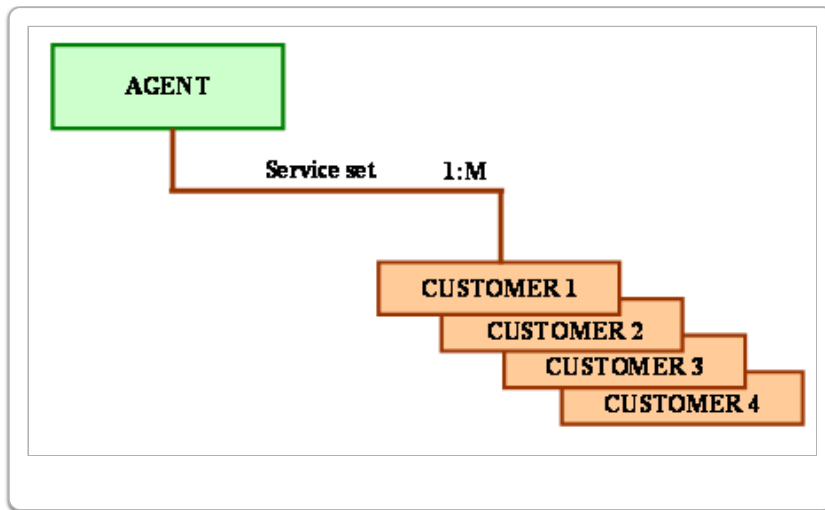
A relational diagram



If the relationship between AGENT and CUSTOMER (one agent has many customers) was implemented in a hierarchical model it would look like this:



If the relationship between AGENT and CUSTOMER were implemented in a network model, it would look like this:



Test Yourself 1.8

Select all that are true of data models. (Check all that apply.)

There are two main categories of data models.

This is false. There are three main categories of data models: conceptual model, physical model, and logical model.

A data model represents data structures and their characteristics, relations, constraints, and transformations.

This is true. A data model provides a good blueprint for database design. The data model has data structures along with their characteristics, relationships, constraints, and transformations.

There is only one standard of data model in database design.

This is false. There are numerous standards of data models in database design. For example: Crow's foot, Chen model, and UML.

A data model represents a simple diagram of complex data structures.

This is true. The data model acts as a diagram to communicate the data and its relationships.

A high-level model which is based on entities and relationships is called the *conceptual* data model.

This is true. The conceptual data model is a high overview model.

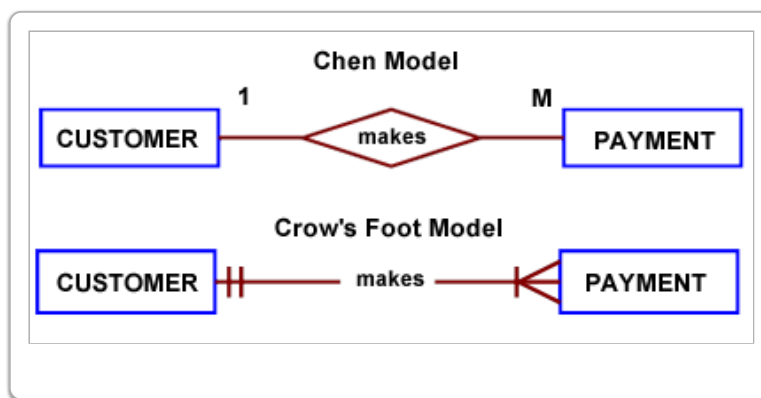
Basic Data-Modeling Building Blocks

The basic building blocks of all data models are entities, attributes, and relationships. An entity is anything, such as a person, place, thing, idea, or event, about which data are to be collected and stored. Entities may be physical objects such as customers or products. Entities may also be abstractions such as flight routes or accounts. An attribute is a characteristic of an entity. For example, a CUSTOMER entity would have attributes such as their last name, first name, phone number, address, and credit

limit. The attributes are the equivalent of fields in file systems. A relationship describes an association among two or more entities. For example, a relationship between customers and agents might be described as “an agent can serve many customers and each customer may be served by one agent.” Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M, M:N, and 1:1 for them, respectively.

An entity-relationship model, also known as an ERM, helps identify the database's main entities and their relationships. Because the ERM components are graphically represented, their role is more easily understood by users and designers. When end users and database designers meet to discuss the data requirements of an enterprise or application the ERM diagrams are a primary means of communicating those requirements and expressing them in a precise way that can be used as a basis for more detailed database design. When users and the database designers have agreed on an ERM that correctly describes the entities and their relationships, the database designer can map the ERM to the relational database model's tables and attributes. This mapping process uses a series of well-defined steps to generate all the required database structures. This structure-mapping approach is augmented by a process known as normalization, which is covered later.

“A customer can make many payments, but each payment is made by only one customer” serves as the basis for the entity relationship diagram (ERD) presentation below.



Business Rules and How They Affect Database Design

A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization's environment. Business rules do not just apply to traditional business, but also to educational institutions, religious groups, government units, and other organizations. Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment. Properly written business rules are used to define entities, attributes, relationships, and constraints. Knowing the business rules promotes the creation of an accurate data model based on how the organization actually works and what role is played by the data within that organization's operations. Consequently, the database designer must identify the organization's business rules and analyze their impact on the nature, role, and scope of data.

A business rule defines or constrains one aspect of your business that is intended to assert business structure or influence the behavior of your business. Business rules often focus on access control issues; for example in a bank the following business rule could apply, “If a customer closes a safety deposit box, the key and lock must be replaced (or switched with an empty box) before the next customer is allowed to use the safety deposit box.” When a business rule is discovered, created or changed it may occasionally require us to redesign the data model. Because databases and the data that they store must persist in spite of

changes to business rules, it is important to design databases so that they are independent of the kinds of business rules that may change. For example, the following are business rules that one would expect to be represented in a database for flight operations, because they are highly unlikely to change, and because they are fundamental to representing a flight segment, which is the term for everything involved in an aircraft taking off as part of a scheduled flight, flying and landing:

- A flight segment has one originating city and one destination city, which are normally different.
- A flight segment has one aircraft.

In contrast, the following hypothetical business rules should probably not be represented in the database structure, because they may change:

- A flight segment has at least three flight attendants.
- Flights from Los Angeles to San Francisco use only Boeing 737 aircraft.

Although many business rules should not be enforced in the database structure, the data stored in the database will be dependent upon and constrained by the business rules. It can be technically difficult and expensive to change the database structure when business rules change. It is therefore important that the database structure support not only the current business rules, but also likely future changes to those business rules.

Test Yourself 1.9

Select all that are true of business rules. (Check all that apply.)

Business rules only apply to traditional businesses.

This is false. Business rules do not just apply to traditional business, but also to educational institutions, religious groups, government units, and other organizations.

Business rules are written to describe only entities or attributes.

This is false. Business rules are used to define entities, attributes, relationships, and constraints.

When a new business rule is discovered it may require a modification to the data model.

This is true. A business is constantly changing and there are always new business rules added. This may require a modification in the current data model.

Data stored in the database will be dependent and constrained by the business rules.

This is true. Since the database is built off of business rules. The data stored will be dependent and constrained correctly.

A good business rule could be: A student may only take one class.

This is true. A good business rule describes the use of entities or relationships between other entities.

Evolution of Major Data Models

The first data model, which was developed in the 1960s, is known as the *hierarchical* model. The hierarchical database is a collection of records that is logically organized to conform to the upside-down tree (hierarchical) structure. Within the hierarchy, the top layer (the root) is perceived as the parent of the segment directly beneath it. While this model represents 1:M relationships, it does not represent M:N relationships.

The next model is known as the *network* model. In network database terminology, a relationship is called a set. Each set is composed of at least two record types: an owner record and a member record. The difference between the hierarchical model and the network model is that the latter might include a condition in which a record can appear (as a member) in more than one set. In other words, a member may have several owners. A set represents a 1:M relationship between the owner and the member. Some of the weaknesses of this model are the structural complexity, the lack of structural independence, and the consequent support for only navigational queries, with no ad hoc query language such as SQL.

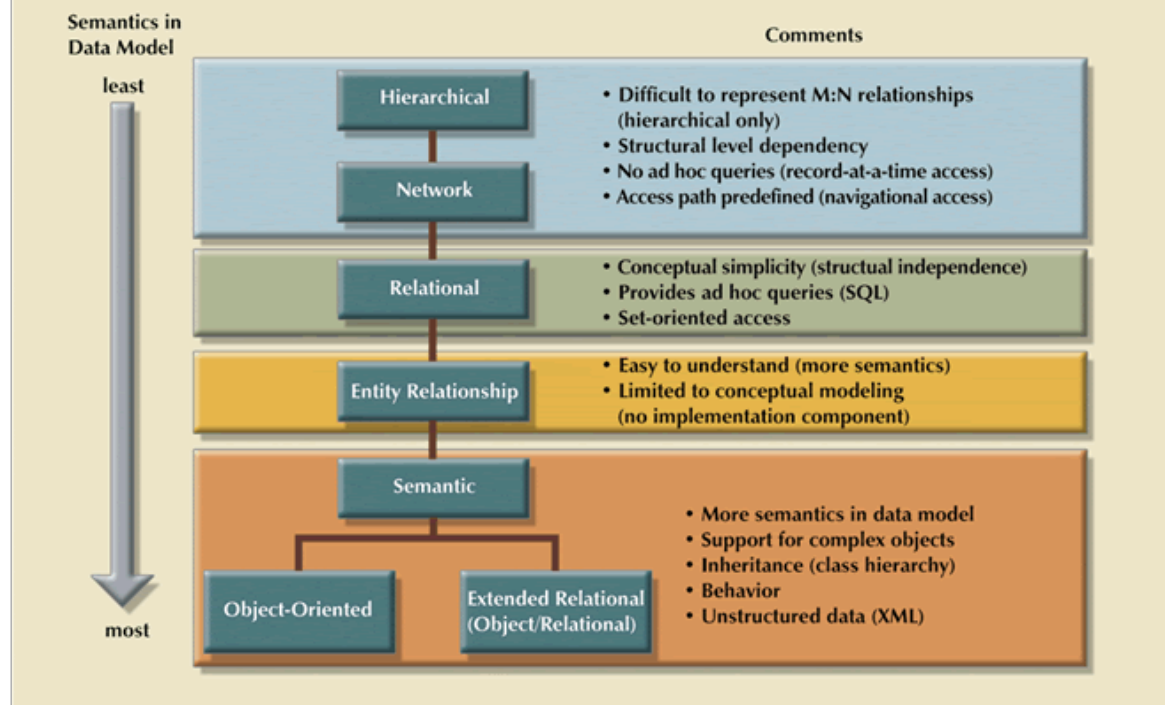
A third model is the *relational* model. The basic building block of the relational model is the *table*, which is a matrix of rows and columns. Tables are related to each other via a common entity characteristic. All three relationship types are easily represented in this model. One disadvantage of the relational model that was more important in the early days of computing is that it requires substantial computer resources to run the Relational DBMS (RDBMS).

An alternate model is the previously mentioned Entity Relationship Model (ERM). In this model, entities are drawn by using diagrams with connectors that depict their relationships. This model has the advantage of visually depicting relationships. Entity-Relationship models are in the family of object-oriented models, which is different than the hierarchical, network and relational models, which are in the family of record-oriented models. Record-oriented models are at best awkward for people to understand and use, so entity-relationship and other object-oriented models are universally used for databases, even when those databases are record-based.

In full-fledged object-oriented models, entities are represented as objects that contain both data and operations. Advantages of this model include the addition of semantic content, the ability to more compactly represent complex domains, and the ability to represent applications software written in object-oriented languages such as Java. A disadvantage is the steeper learning curve. The traditional entity-relationship model and the most important features of object-oriented models have been combined in the extended (or enhanced) entity-relationship model (EERM), which we will study in Lecture 8.

The following diagram describes the evolution of major data models-hierarchical, network, relational, ER and then finally Semantic. Their advantages and disadvantages are noted in the comments.

FIGURE 2.8 The development of data models



Advanced Topic: The fundamental difference between the relational and entity-relationship data models.

Technically the entity-relationship and extended entity-relationship models are in the broad class of object-based models, because they represent data as objects and their relationships. The network, hierarchical, and relational models are in the broad class of record-based models, which represent data as collections of associated records. This difference makes the process of mapping an entity-relationship model to a relational database nontrivial. For example, entity-relationship and object-oriented models can represent many to many relationships, which cannot be represented directly in record-based models. There are many reasons why entity-relationship, extended entity-relationship, and object-oriented models are used to model data that will be implemented in relational databases, but the main reason is that record-oriented models do not have the expressive power needed to serve as an effective way to express and communicate data requirements. The details of mapping the more easily understood object-based models to the record-oriented models is left to the database designer.

Test Yourself 1.10

Select all that are true of the various aspects of data modeling. (Check all that apply.)

The first data model developed was the *network model*.

This is false. The first data model developed was the hierarchical model.

The basic building block of the relational model is a table.

This is true. A matrix of rows and columns, which is a table, is the basic building block of a relational model.

The entity-relationship model entities are drawn by using diagrams that show their relationships.

This is true. The main purpose of the entity-relationship model is to show the relationships between the entities.

The traditional entity relationship model and object-oriented model have been combined to create the extended entity relationship model.

This is true. The two models have combined to create the EERM, to show more advanced details of the intended rules.

Object oriented models are easier to understand than entity relationship models.

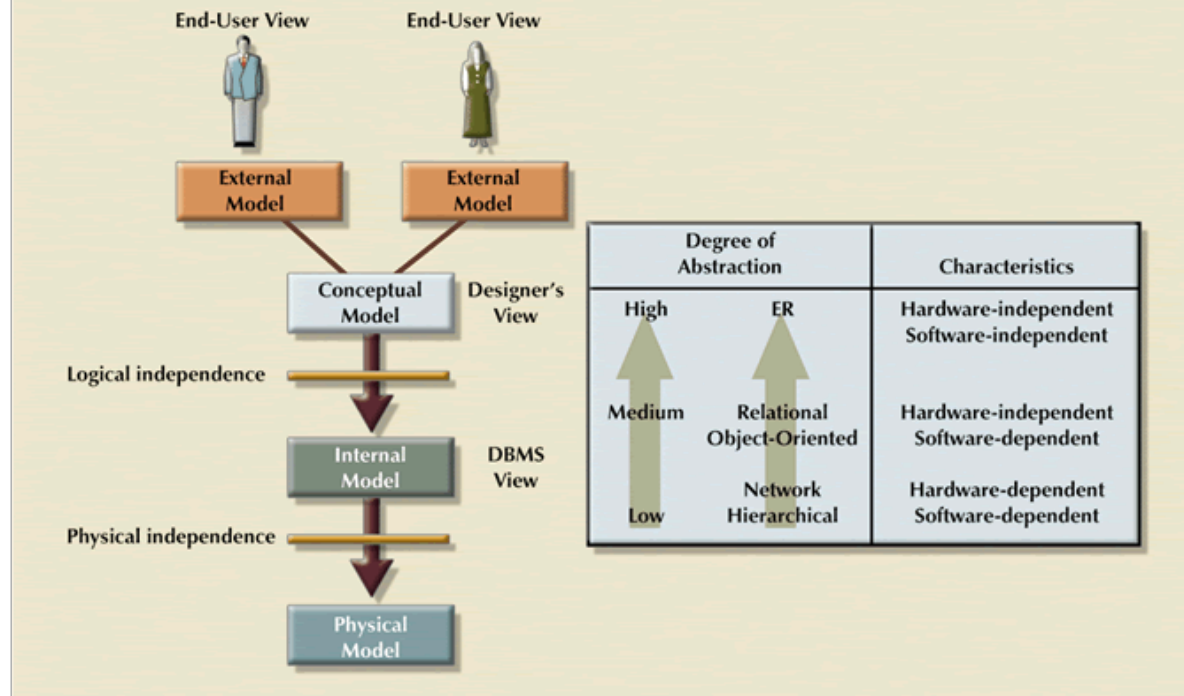
This is false. Entity relationship models are less complex than object-oriented models.

Classifying Data Models by Level of Abstraction

When designing a database, the designer starts with an abstract view of the overall data environment and adds details as the design approaches implementation. The design of a database can be divided into four models with decreasing levels of abstraction. The *conceptual model* represents a global view of the database. It is the basis for the identification and description of the main data objects, avoiding details. The most widely used conceptual model is the entity relationship (ER) model. Once a specific DBMS has been selected, the *logical model* adapts the conceptual model to the DBMS. The internal model is the representation of the database as "seen" by the DBMS. The external model, based on the internal model, is the end user's view of the data environment. The physical model operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices.

Representational or implementation data models are the models used most frequently in traditional commercial DBMSs, and they include the widely-used relational data model, as well as the so-called legacy data models-the network and hierarchical models-that had been widely used in the past. Object-oriented models and object-oriented databases which directly support them have been used for years in semantically-rich problem domains such as computer aided design and manufacturing. Modern DBMS such as Oracle and IBM's UDB, as well as the ANSI SQL 2003 standards, include extensions to the relational model to support object-oriented features. These object-relational standards and the object-relational database management systems (ORDBMS) that implement them represent a synergistic synthesis of database technologies. For the data modeler this means that the DBMS can more directly represent their data models. For the database designer these new database technologies open many new design opportunities and additional design complexity, along with opportunities for greatly improving database performance and the opportunity to more straightforwardly represent semantically rich data directly in the database.

FIGURE 2.9 Data abstraction levels



Test Yourself 1.11

Select all that are true of the different levels of abstraction. (Check all that apply.)

The external model is the highest level of data abstraction.

This is true. The external model is the end-users view of the data.

The data model is not a level of data abstraction.

This is true. The data model is the model that has levels of abstraction in it.

The conceptual model is a level of data abstraction and represents the global view.

This is true. The conceptual model is the most widely used model and it shows the overall global view.

The logical is the lowest level of data abstraction and is mainly concerned with storage methods.

This is false. The lowest level of data abstraction is the physical model, which is mainly concerned with storage methods.

The physical model maps the conceptual model to the DBMS.

This is false. The internal model maps the conceptual model to the DBMS.

Summary

In this lecture we have learned about the critical role of data modeling in discovering and communicating data requirements. We learned how business rules describe business processes and constraints and learned that we can translate business rules into data models. We learned about the historic evolution of data models, and that entity-relationship modeling is the modeling technique most frequently used for business databases. We will study entity-relationship modeling in Lecture 4. We learned that normalization is a sequence of tests that we apply to entities in our models, to eliminate redundancy and the potential for data anomalies. We will study normalization in Lecture 5. We learned that entity-relationship modeling has been extended with semantic (object-oriented) modeling constructs to form extended (or enhanced) entity-relationship models (EERM), which combine the object-oriented concepts of generalization and specialization with the relational concepts of keys. We will study EERM in Lecture 6. We also learned that mainstream commercial object-relational DBMS such as Oracle and IBM's UDB and the ANSI SQL 2003 standards include both relational and semantic (object-oriented) features. We will learn how to model these semantic features using the Extended Entity-Relationship Model in Lecture 8, and learn how to use the object-oriented extensions in object-relational DBMS in CS779 Advanced Database Management.

■ Lecture 3 - Introduction to SQL

Introduction to the Structured Query Language (SQL)

It is important for a database language to perform several basic functions. First, it must enable the creation of database and table structures. Second, it must allow users to perform basic data management chores such as adding, deleting, and modifying data. Third, it must be able to perform complex queries and transform raw data into useful information. Structured query language (SQL) meets these requirements fully. It provides facilities for meeting basic database needs while maintaining an easy-to-use syntax. SQL functions fit into two main categories. These are: a data definition language with commands to create database table structures and to define access rights to the database; and a data manipulation language with commands to update, delete, and retrieve data within the database tables. SQL is a nonprocedural language with a basic vocabulary set of less than 100 words.

The SQL language is comprised of:

- 1. DDL:** data definition language. DDL is used to define the tables that make up the database. For example, the "Create Table" syntax.
- 2. DML:** data manipulation language. DML is concerned with the retrieval and update of the database. For example, the "Select * from" syntax.
- 3. DCL:** data control language. DCL is used to specify who can access data in the database and what operations they can perform. For example, the GRANT and REVOKE commands are part of DCL.

We have introduced the relational model. A relational database consists of tables. Tables, of course, are a very simple data structure. Every database system requires a language for accessing data; the language discussed in this course is SQL.

Test Yourself 1.12

Select all that are true of the Structured Query Language (SQL). (Check all that apply.)

The data definition language is concerned with the retrieval and update of the database.

This is false. The data definition language (DDL) is used to define/create structures in the database.

The data manipulation language is used to define tables and other structures that make up the database.

This is false. The data manipulation language (DML) is used to retrieve or update data within the database.

SQL is very complex and has an extensive vocabulary set of more than 1,000 words.

This is false. SQL is a simple, nonprocedural language with a vocabulary under 100 keywords.

The data control language (DCL) is used to grant and revoke privileges that different users have on the database and data in the database.

This is true. The data control language (DCL) is used to specify security measures, on who can access data in the database.

An example of a DML statement is: 'SELECT * FROM'.

This is true. DML is used to retrieve data. SELECT * FROM, is a good example.

SQL and Data Administration

Before a DBMS can be used, you must complete two tasks. First, you create the database structure to hold the tables. For the DBMS, this structure is a set of physical files on disk, called a schema, and is created by using a CREATE statement. An example is CREATE SCHEMA AUTHORIZATION. When the new database is created, the DBMS automatically produces the tables that hold the metadata. After developing the structures, define the tables for end user data. Again, use the CREATE statement. It takes the general form CREATE TABLE.

The database consists of tables. Each table is a two dimensional structure of rows and columns. A row is sometimes referred to as a *record* and a column is sometimes referred to as a *field*; these terms are carried over from file-based data processing. The DDL is used to define (create) tables, views, and indexes.

We create tables in accordance with our data management needs; as our needs grow it may be necessary to add new fields to existing tables. It may also be necessary to add new tables.

Test Yourself 1.13

Select all that are true of data administration. (Check all that apply.)

A schema is the database structure that holds the tables in physical files on a disk.

This is true. The database schema is used to describe the structure of the database.

The database automatically creates the metadata tables.

This is true. When the database is created, the metadata tables are created automatically.

To create database tables you must use DML commands.

This is false. To create database tables you must use DDL commands. DML is used for retrieving data.

It is not possible to create new tables once the database is in use.

This is false. New tables are created all of time. It is typical to create new tables all of the time, even when the database is in use.

The Basics of Data Manipulation in a Database

This section shows you the basics of manipulating data in a relational database step-by-step. Read it carefully. It is highly recommended that you execute the commands presented in this section in your own database, as there is no substitute for your own hands-on experience. We first discuss creating a storage container which is capable of holding many rows of data. We then discuss the commands to add and manipulate the data. Lastly we describe how to remove the storage container from the database should the need arise.

The primary storage container in a relational database is the table. A table is perceived as a two-dimensional structure composed of rows and columns. The columns of a table collectively define the kind of data that may be stored in the table, and each row in a table is a particular set of values for all of the columns in that table. The definition of each table includes at least one column, and the definition of each column includes at a minimum a column name and a datatype. The datatype determines the legal value for the column.

For an example to help you visualize a relational table, consider the following spreadsheet:

Food Item	Purchase Date	Price
Sandwich	25-Aug-2011	4.15
Eggs	1-Sep-2011	3.00
Pizza	2-Sep-2011	10.50
Apple	2-Sep-2011	1.00

Intuitively we see that there are three columns in this spreadsheet, named "Food Item", "Date Bought", and "Price". Next, we see that each column has an expected kind of value. The food items are a series of characters, the dates bought are in a date format, and the prices are in a numeric format. We also see that there are four rows in this spreadsheet. Lastly, the spreadsheet is a two-dimensional structure composed of rows and columns.

This spreadsheet could easily be stored in a relational database table. We would create three columns with similar names, and assign the appropriate datatypes to each column. We would then insert the data into the table.

Which of the following are true of database tables? (Check all that apply.)

Relational database tables are one dimensional

This is false. Relational database tables are perceived as two-dimensional structures. One dimension is the available columns in the table, and the second dimension is the rows in the table.

A relational table row has a datatype that defines the legal values for that row

This is false. While each column in a relational table has a datatype, each row does not. A row is one set of values for the columns present in a relational table.

Each table column in a relational table has a name

This is true. Each column in a relational table has a name that is distinct from any other column name in the same table. Each column is identified by its name.

Data in a two-dimensional spreadsheet, such as an Excel spreadsheet, can usually be stored in a relational database table

This is true. Relational tables are also two-dimensional, and most any data we can store in a spreadsheet, we can store in a relational table.

Creating the Container to Hold the Data

Below is an example of a SQL command which creates a table similar to the spreadsheet.

```
CREATE TABLE Food_purchase (  
    food_item VARCHAR(64),  
    purchase_date DATE,  
    price DECIMAL(4,2)  
);
```

Let us analyze the command piece by piece. The command begins with the SQL keywords CREATE TABLE. A SQL keyword is a word that the designers of the SQL language chose to carry special meaning within the language. The two keywords CREATE TABLE together indicate to the SQL compiler that we are beginning a command to create a table. The next word, "Food_purchase", is the name of the table we are creating. This is not a keyword, but simply an identifier of our choosing. We could have chosen an alternative identifier, "Purchase_of_food", for example. SQL compilers know that by definition, the identifier following the CREATE TABLE keywords defines the name of the table.

Only certain characters are legal in identifiers, and the legal characters depend upon the particular database we are using. Probably the most common characters used are letters, numbers, and the underscore. Although a space is legal in an identifier in many databases, its use is not recommended, because then the identifier must always be quoted whenever it is referenced, and treated specially in certain situations. One common substitute for the space is the underscore character, as you observed in the "Food_purchase" table.

Next let us examine the clause that begins and ends with parentheses. The left parenthesis begins the specification of the columns and constraints for the table, while the right parenthesis closes the same specification. We will learn about constraints

later. For now, we focus on the column specifications. The first thing you may notice is that each column specification is separated by a comma, and the last specification in the list has no comma. We would put fewer specifications if we had fewer columns, and more specifications for more columns. The next thing you may notice is that each column specification has two words.

Although each column specification may have more than two words that define additional aspects of the column, it must have at a minimum both the column's name and the column's datatype. The first word is an identifier specifying the name of the column. Just as with the table name, this is not a SQL keyword, but an identifier, and we can choose a variety of names. A good identifier describes well what it represents. The second word in a column specification is a SQL keyword indicating the column's datatype. Some datatype keywords are followed by additional constraints on the datatype.

Now that we have discussed some parts of a column specification, let us examine the first in the Food_purchase table definition, "food_item VARCHAR(64)". The name of the column is "food_item" and its datatype is "VARCHAR(64)". The VARCHAR keyword indicates that the column will store a variable number of characters, for example, "Pizza" or "Eggs". The "(64)" clause constrains the number of bytes in the column to a maximum of 64. An attempt to store more than 64 bytes in the column will result in an error. The maximum number of characters *must* be specified for VARCHAR columns. The legal maximum depends upon the database. As of Oracle 11g Release 2, Oracle does not allow more than 4,000 bytes for a VARCHAR column. As of SQL Server 2008 R2, SQL Server does not allow more than 8,000 bytes for a VARCHAR column.

We now examine the second column specification, "purchase_date DATE". The column has a name of "purchase_date", and a DATE datatype. A DATE datatype indicates that a year, month, and day may be stored in the column. Some database management systems also allow additional time information to be stored in a DATE column. For example, Oracle allows the additional storage of hours, minutes, and seconds in a DATE column. However, the standards for SQL specify that DATE columns only store the year, month, and day, and it is a best practice to use DATE columns to store only these fields, for portability.

The last column, described by the "price DECIMAL(4,2)" specification, has a name of "price" and a datatype of DECIMAL. A column with a DECIMAL datatype can store numbers with or without digits after the decimal point. In this example, the "(4,2)" constrains the maximum number of digits before the decimal point to 4, and the maximum number of digits after the decimal point to 2. Therefore, 10.95 could legally be stored in the price column, but 10,000 could not, and 3.4562 could not.

Test Yourself 1.15

Which of the following are true of database tables? (Check all that apply.)

The SQL command to create a table begins with TABLE CREATE

This is false. The SQL command to create a table begins with CREATE TABLE.

All relational table names must begin with the word "table"

This is false. The identifier for a table name can be most anything of our choosing, and need not begin with the word "table".

A table name can consist of any character available

This is false. Databases restrict the legal characters for a table name. Probably the most commonly used characters are letters, numbers, and the underscore.

A column definition begins with the datatype declaration

This is false. A column definition begins with the name of the column. The datatype comes after.

A numeric datatype for a column can be restricted to a maximum number of digits, and a maximum number of decimal digits

This is true. The keyword DECIMAL, and other similar numeric types, allow a maximum for both digits and decimal digits.

Adding the Data

The next thing we need to do is to add the data into our table. The technical SQL term used for adding data is *insert*, and the corresponding SQL keyword is, unsurprisingly, INSERT. Let's insert the first row in our spreadsheet:

Sandwich	25-Aug-2011	4.15
----------	-------------	------

We do so with the following command:

```
INSERT INTO Food_purchase (food_item, purchase_date, price)
VALUES ('Sandwich', CAST('25-Aug-2011' AS DATE), 4.15);
```

The SQL keywords INSERT INTO together indicate to the SQL compiler that we are beginning a command to add data into a table. The next word, "Food_purchase", identifies the table we are adding data to. Unlike with the CREATE TABLE command, we cannot type any identifier we please; rather, we must type the name of our table exactly.

You will notice that there are two comma-separated lists within parentheses. The second list clearly contains the data we are inserting. So what is the first list? It specifies the column names in the Food_purchase table corresponding to each data item in the second list, in the same order as the data. Thus, the Sandwich value is to be inserted into the food_item column, Aug-25-2011 is to be inserted into the purchase_date column, and 4.15 is to be inserted into the price column. Both the ordering and the number of items in both lists must be the same. The VALUES keyword tells the SQL compiler that the next comma separated list contains the values to be inserted. Though it is possible to omit the first comma separated list by inserting the values in the order they exist in the database, it is not recommended to do so. Production strength SQL insertions specify the column names as illustrated, to help prevent several cases where data is unknowingly inserted into the wrong column.

With a little thought, you can probably guess what commands we would need to type to insert the last three rows in our spreadsheet.

Eggs	1-Sep-2011	3.00
Pizza	2-Sep-2011	10.50
Apple	2-Sep-2011	1.00

```
INSERT INTO Food_purchase (food_item, purchase_date, price)
VALUES ('Eggs', CAST('1-Sep-2011' AS DATE), 3.00);
```

```
INSERT INTO Food_purchase (food_item, purchase_date, price)
VALUES ('Pizza', CAST('2-Sep-2011' AS DATE), 10.50);

INSERT INTO Food_purchase (food_item, purchase_date, price)
VALUES ('Apple', CAST('2-Sep-2011' AS DATE), 1.00);
```

Test Yourself 1.16

Which of the following are true of adding data using SQL? (Check all that apply.)

The SQL command used to add data to tables is INSERT INTO.

This is true. What we casually phrase "adding data" is accomplished in SQL by "inserting into" tables.

A single INSERT INTO command in SQL can add data to multiples tables.

This is false. A single INSERT INTO command adds data to only one table.

If two comma separated lists are present in an INSERT INTO command, the first comma separated list contains the values to be inserted into each column.

This is false. If two comma separated lists are present in an INSERT INTO command, the first comma separated list contains the names of the columns into which the values will be inserted. The second comma separated list contains the values to be inserted into each column.

The name of the table into which data will be inserted must be specified in an INSERT INTO command.

This is true. Without the table name, an INSERT INTO command would be syntactically invalid. The SQL engine needs to know into which table to insert the data.

If there is a Food_item table with two columns named "food_name" and "price", where "food_name" contains a series of characters and "price" contains a numeric value with two decimal points, a command to insert data into this table could be:

```
INSERT INTO Food_item (food_name, price)
VALUES (3.25, 'Carrots');
```

This is false. All items in the example command are correct, except for the values, which have been reversed.

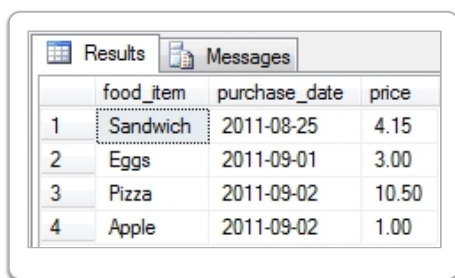
Viewing the Data

Since we have created our table and inserted the data, the next logical step is to view the data we have inserted. Doing so is quite simple once we understand how to do so. The technical word for retrieving data in SQL is *selecting*, and the SQL keyword associated with this command is, unsurprisingly, SELECT. A command to view all data in our Food_purchase table is:

```
SELECT food_item, purchase_date, price
FROM Food_purchase;
```

The SELECT keyword indicates to the SQL compiler that we are beginning a command to retrieve data. Next, the comma separated list of column names indicates which columns' values we will see. In this case, we have specified all three columns in the Food_purchase table. The FROM keyword, when used as part of a SELECT statement, indicates to the SQL compiler that we will next specify the tables or tables from which the data will be retrieved. In this case, we have told the compiler that we want to see the data in the Food_purchase table, simply by listing the table's name after the FROM keyword.

The way SQL results are shown depends upon the SQL client used, and different clients show the results differently. However, virtually all SQL clients will show the results of a SELECT statement with the column names in the highest horizontal row on the screen, followed horizontally by the rows of data. The screenshot below illustrates the results of the SELECT statement above in Microsoft SQL Server Management Studio. The appearance of the results will differ from client to client, and from DBMS to DBMS, but the results will have the same tabular format.



	food_item	purchase_date	price
1	Sandwich	2011-08-25	4.15
2	Eggs	2011-09-01	3.00
3	Pizza	2011-09-02	10.50
4	Apple	2011-09-02	1.00

Advanced Topic: The Semicolon

You may have noticed that semicolons are sometimes present at the end of SQL statements. The semicolon is not really part of a SQL statement; rather, it is a separator between statements. In SQL clients, when you want to execute more than one SQL statement at a time, the semicolon is required. Many SQL clients will execute a single SQL statement without the presence of a semicolon. In some interactive SQL systems, the semicolon serves as a signal to the parser that you have completed a statement that is ready to be interpretively executed.

Test Yourself 1.17

Which of the following are true of viewing data using SQL? (Check all that apply.)

The SQL command used to view data in tables is RETRIEVE.

This is false. The SQL command used to view data in tables is SELECT.

You may specify the columns you would like to see when using the SELECT command in SQL.

This is true. The column names are comma-separated immediately after the SELECT keyword.

There is no need to specify the name of a table when using the SELECT command, because the SQL engine can determine the table from the column names.

This is false. The table name must be specified in the SELECT command. Different tables can have columns with the same name.

The SELECT Command

The SELECT command is primarily used to retrieve data from the database. It is also used when creating a copy of a table or creating views, and can be used to specify rows for updating. In this chapter we concentrate on its use for retrieving data.

The basic form of the **SELECT** command is:

SELECT field-list

FROM table-list

WHERE field-expression

GROUP BY group-fields

HAVING group-expression

ORDER BY field-list

The result of the SELECT is a listing of data derived from some set of tables in the database. The "field-list" specifies the fields to be listed, such as USERID, NAME, AGE. The data listed is obtained from the set of tables (table-list) specified in the FROM clause. The "field-expression" in the WHERE clause specifies a Boolean expression that rows in the "table-list" must satisfy to be included in the listing. The GROUP BY clause is used when we wish to summarize information in the underlying tables. For example, GROUP BY subject causes the rows to be organized into groups, one group for each unique value of the subject field. The HAVING clause is used to specify which groups are to be included. ORDER BY is used to sequence the rows of the listing. The WHERE, GROUP BY, HAVING, and ORDER BY clauses are optional.

All queries in SQL are based on the SELECT command. The SELECT * FROM TABLE command lists all rows for all columns in a particular table. This query can be modified to restrict the search to particular desired data. Use an optional WHERE clause with the SELECT command to retrieve rows that match specific criteria. The WHERE clause combines values with mathematical comparison operators (=, <, <=, >, >=, <>) and logical operators (AND, OR, NOT) to enable the user to specifically define a desired output. The SELECT command also can create computed columns with an assigned alias, or alternative name, to make them more readable. Some versions of SQL also include special operators (BETWEEN, IS NULL, LIKE, IN, EXISTS) to enable complex query operations with reduced programming effort. Many SELECT statements require that data is retrieved from more than one source table. To join tables, use the FROM clause of the SELECT command. To get correct results, choose only rows in with common attribute values.

SELECTing from a Single Table

We continue showing you the SQL to implement the relational operations involved in selecting data. These are simple examples; we will learn more about SELECT in Chapter and Lecture 7. We begin with Figure 3.9 from the text:

FIGURE 3.9 **SELECT**

Original table

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

SELECT ALL yields

New table or list

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

SELECT only PRICE less than \$2.00 yields

P_CODE	P_DESCRIPT	PRICE
213345	9v battery	1.92
254467	100W bulb	1.47

SELECT only P_CODE = 311452 yields

P_CODE	P_DESCRIPT	PRICE
311452	Powerdrill	34.99

The SQL to create this Product table is:

```
CREATE TABLE Product(  
  p_code DECIMAL(6),  
  p_descript VARCHAR(30),  
  price DECIMAL(9,2)  
)
```

The SQL to select (project) all of the data from the Product table is:

```
SELECT * FROM Product;
```

The asterisk after SELECT means to select all columns.

We now consider restricting the rows. The SQL to select the rows from Product where the price is less than \$2.00 is:

```
SELECT * FROM Product  
WHERE price < 2;
```

The SQL to select the rows from Product where the p_code = 311452 is:

```
SELECT * FROM Product  
WHERE p_code = 311452;
```

If this seems simple, it's because it is. We continue with Figure 3.10 from the text:

FIGURE 3.10

PROJECT

Original table

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PROJECT PRICE yields

New table or list

PRICE
5.26
25.15
10.99
1.92
1.47
34.99

PROJECT P_DESCRIPT and PRICE yields

P_DESCRIPT	PRICE
Flashlight	5.26
Lamp	25.15
Box Fan	10.99
9v battery	1.92
100W bulb	1.47
Powerdrill	34.99

PROJECT P_CODE and PRICE yields

P_CODE	PRICE
123456	5.26
123457	25.15
123458	10.99
213345	1.92
254467	1.47
311452	34.99

The SQL to project only the price column from the Product table is:

```
SELECT price
FROM Product;
```

The SQL to project the p_descript and price is:

```
SELECT p_descript, price
FROM Product;
```

The SQL to project the p_code and price is:

```
SELECT p_code, price
FROM Product;
```

We can easily combine restriction and projection. For example, to project the p_descript and price from products with a price greater than \$10 the SQL is:

```
SELECT p_descript, price
FROM Product
WHERE price >10;
```

Advanced Topic: Real DBMS don't begin by computing the Cartesian product

At this point our text says that the first step in joining tables is to compute the Cartesian product of the tables. This is unfortunately misleading. The DBMS avoids computing the Cartesian product, because the Cartesian product can be very large and very expensive to compute. For example, I have joined two ten billion row tables and obtained the results within seconds. The Cartesian product of two ten billion row tables would have 100 billion billion rows. Even with a short row length of ten bytes per row

this would require over a billion terabytes of storage. Storing the Cartesian product would require about a billion large disk drives. Computing the Cartesian product is clearly not viable. Real DBMS start with the restrictions (the WHERE clauses). That said, joins performed by starting with restrictions produce the same results as joins performed by computing the Cartesian product and then projecting and restrict it. This is one reason why it is essential to understand the mechanics of performing a Cartesian product.

Test Yourself 1.18

Which of the following are correct? (Please check all of the following that are correct.)

The SQL statement "SELECT * FROM Pet;" would retrieve all rows and all columns from a table named Pet.

This is true. The asterisk indicates all columns. Since there is no WHERE clause, all rows would be returned.

The SQL statement "SELECT pet_species FROM PET;" would project only the pet_species column from the Pet table.

This is true. The result would have the pet_species column (only) for all the rows in the Pet table.

The SQL statement "SELECT * FROM Pet WHERE pet_species = 'CAT';" would retrieve only the pet_species column from the Pet table for rows where the value of pet_species is 'CAT'.

This is false. The given statement would retrieve all columns for only the rows in the Pet table with a value of "CAT" in the pet_species column.

The SQL statement

```
SELECT pet_species FROM Pet WHERE pet_name = 'Fido'
```

would retrieve only the pet_species column from the Pet table for rows where the value of pet_name is 'Fido'.

This is true. This statement combines restriction and projection. It projects only the pet_species column from rows restricted to those where the value of pet_name is 'Fido'. (If any rows would actually meet this criteria)

In reality, the first step in all joins of relational tables is the computation of the Cartesian product.

This is false. This is not true. Real DBMS start with the restrictions (the WHERE clauses). Because of the potential size of the Cartesian product, computing it could be very expensive and require huge storage capacity. The Cartesian product of a practical multi-table join can have more rows than the number of atoms in the universe.

Removing Data

Sometimes we decide that a row of data in the table is no longer relevant or correct, and want to remove the row. Doing so is not difficult; we only need execute a command that identifies that we want to delete a row, and that also identifies the specific row to be deleted. The technical word for removing data in SQL is *delete*, and the SQL keyword is not surprisingly, DELETE. We can delete the second row in the Food_purchase table by executing the following command:

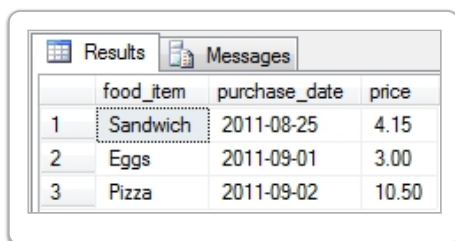
```
DELETE FROM Food_purchase
WHERE food_item = 'Apple';
```

The DELETE FROM keywords together indicate to the SQL compiler that we are beginning a command to delete one or more rows from a table. The identifier following the FROM keyword, in this case, Food_purchase, indicates that we are deleting the rows from the Food_purchase table. The WHERE keyword, when used as part of a DELETE FROM command, indicates that the next (Boolean) expression identifies the rows to be deleted. In layman's terms, a *boolean expression* is a combination of identifiers, literals, and operators that, when evaluated, yields only one of two results—true or false.

The details of Boolean expressions are complicated; however, we can begin to tackle the subject by analyzing the simple expression above. The entire expression is food_item = 'Apple', and the first word food_item is an identifier that identifies the name of a column in the Food_purchase table. The = symbol is a Boolean operator which evaluates whether the results of the expression to its left, and the expression to its right, are equal. The operator yields a true value if they are equal, or false value otherwise. In this case, the expression to its left is simply the food_item column. The expression to its right, 'Apple', is a literal series of characters. A *literal* is a word that represents a static value, and can be contrasted with an identifier, which acts as a reference to a value. In this case, 'Apple' is a representation of the literal series of characters A-p-p-l-e.

Now that we've covered the details of the expression, we can deduce its meaning. The expression is stating, "Yield true if the value in the food_item column is the literal series of characters A-p-p-l-e, and yield false otherwise". When taken in context of the surrounding DELETE FROM command, the database engine would thus only delete the rows where the food_item column matches the A-p-p-l-e series of characters. Conceptually, the database engine retrieves the food_item value from each row, evaluate the expression, and delete the rows when the expression evaluates to true. In the case of our Food_purchase table, only the fourth row matches, and so only the fourth row is deleted.

If we view the data in the table at this point, we see that there are only three rows, because we have deleted the fourth row.



The screenshot shows a database interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with the following data:

	food_item	purchase_date	price
1	Sandwich	2011-08-25	4.15
2	Eggs	2011-09-01	3.00
3	Pizza	2011-09-02	10.50

Modifying Existing Data

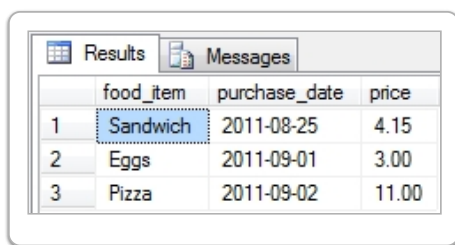
Sometimes we do not want to delete existing data, but want to modify an aspect of it. For example, if we are storing a person's name that has changed, we would want to update the database with the change. It is not a difficult task; SQL makes it easy. The technical word for modifying data in SQL is *updating*, and the SQL keyword is thus UPDATE.

Imagine that the price of pizza increased from \$10.50 to \$11.00. We can update this data in the database by using the following command:

```
UPDATE Food_purchase
SET price = 11.00
WHERE food_item = 'Pizza';
```

After viewing several commands, you may now recognize why we typed the command in this way. The UPDATE keyword indicates to the SQL compiler that we want to update data in a table, and the identifier following that keyword indicates the table to be updated. In this case, the Food_purchase table will be updated. The SET keyword as part of an UPDATE statement indicates to the compiler that a comma separated list of column/value assignments will be provided. In this case, we only wanted to update the price column, and so the phrase price = 11.00 indicates to the compiler to set the price column's value to 11.00. Lastly, you will recognize that the WHERE clause is of the same format as the WHERE clause in a DELETE statement, and follows the same set of rules. In this case, the WHERE clause indicates that the only row to be updated in the Food_purchase table is that for the "Pizza" food item.

After executing this command, you'll notice that the price of the pizza did in fact increase.



	food_item	purchase_date	price
1	Sandwich	2011-08-25	4.15
2	Eggs	2011-09-01	3.00
3	Pizza	2011-09-02	11.00

Deleting Tables

Sometimes we decide that an entire table (storage container) is no longer needed, and we want to get rid of it. The command to do this in SQL is perhaps one of the simplest. When we are getting rid of a table, we are *dropping* the table. To delete our Food_purchase table, we use the following command:

```
DROP TABLE Food_purchase;
```

The DROP TABLE keywords together indicate to the SQL compiler that we are getting rid of a table. The next identifier, in this case "Food_purchase", indicates the table to be dropped. Execute this command, and the Food_purchase table is gone. It's as simple as that! This command should be used with care, because all of the data in the table will also be dropped, and the data cannot always be recovered if you later decide that you want to keep the data.

Summary

Congratulations! You have now learned the basics of manipulating data in a database. Although there are many more details to learn, the SQL commands you have learned in this section will get you a long way, and you are well on your way to using a database to store data in the real world. In fact, database administrators, database developers, end users, and applications regularly use these kinds of SQL statements every day to manipulate data. Onward to more learning!

