Chapter 2
Database Modeling and Design

I. Relational Database Model

Dr. Eng. Shady Aly

Databases

- Three types of databases: Hierarchical هرمي, network شبكي, and relational علائقي database systems

- ***The Relational model (النموذج العلائقي)*** is the foundation of modern database management systems (DBMS) نظام إدارة قواعد البيانات

Databases

- A database is a computerized filing cabinet (خزانة), that stores data defined and "filed" by users within the organization

- Databases are essential component of any information system.

Databases

- The database system has both hardware and software components

- Hardware is the physical storage medium for the data (hard disk, CD, tape, etc.)

- The software is the medium through which the user accesses the physically stored data. This software is called the ***database management system (DBMS)نظام إدارة قواعد البيانات***.

The database management system (DBMS)

- *The DBMS allows the user to store, retrieve, and update data…..*

- There are **three classes of database systems** with different levels of complexity:
  - **Enterprise** databases
  - **Workstation** databases
  - **Personal** databases

DBMS: Enterprise database
قاعدة البيانات المؤسسية

- *a large database that runs on* ***one or* more servers** and may have several remote client users

- It must be capable of handling a large quantity of transactions and the execution must be in real-time في حينه(For example, a transaction involving an ATM debit recorded in seconds)

- DBMS like Oracle (Oracle Corporation) and DB2 **(IBM)** are typically used for these applications

DBMS: Workstation/workgroup database

- Runs on **one server** and distributes information to several client machines running on the same local area network (LAN).

- The DBMS must be capable of handling multiple clients who are independently generating transactions that change the contents of one or more databases running concurrently on the DBMS.

- Microsoft’s SQL Server, which supports client-server architecture, is a popular choice for workgroup applications.

DBMS: personal database

- A personal database runs on a single personal computer (PC)

- The Access DBMS is a good example of a personal database.

The relational database (RDB)

- The relational database uses the concepts of ***attribute****,* ***domain****,* ***relation****,* ***tuple****, and* ***primary key***

- An **attribute** is a name, or label, for a set of data. “**employee_last_name**”

- A ***domain*** is the set of possible value of the attribute in the database

if the enterprise has three employees

(Joseph Smith, John Doe, and Mary Murphy),

then the three values “Smith,” “Doe,” and “Murphy” are the domain of the attribute “employee_last_name.”

RDB

- A ***relation*** refers to a set of related attributes as defined by a user.

  “employee_SS_no,” “employee_last_name,” and “employee_first_name.”

- A ***tuple*** is a set of related data values from within a relation.

036-27-5192, Smith, Joseph

357-19-9921, Doe, John

142-36-1529, Murphy, Mary

RDB

- A ***primary key*** is an attribute in which the domain value is unique (i.e., not repeated in any tuple of the relation).

employee_SS_no corresponds to a primary key.

RDB

- Relational model allows the user to view the data in a simple intuitive tabular structure, called a ***table***

- A ***table*** is a logical view of related data. The table is defined by the entity set and attributes that it represents

- An ***entity*** is a person, place, event, concept, or thing about which information is needed

- A related group of entities, the information about which is maintained in the same table, is called an ***entity set***

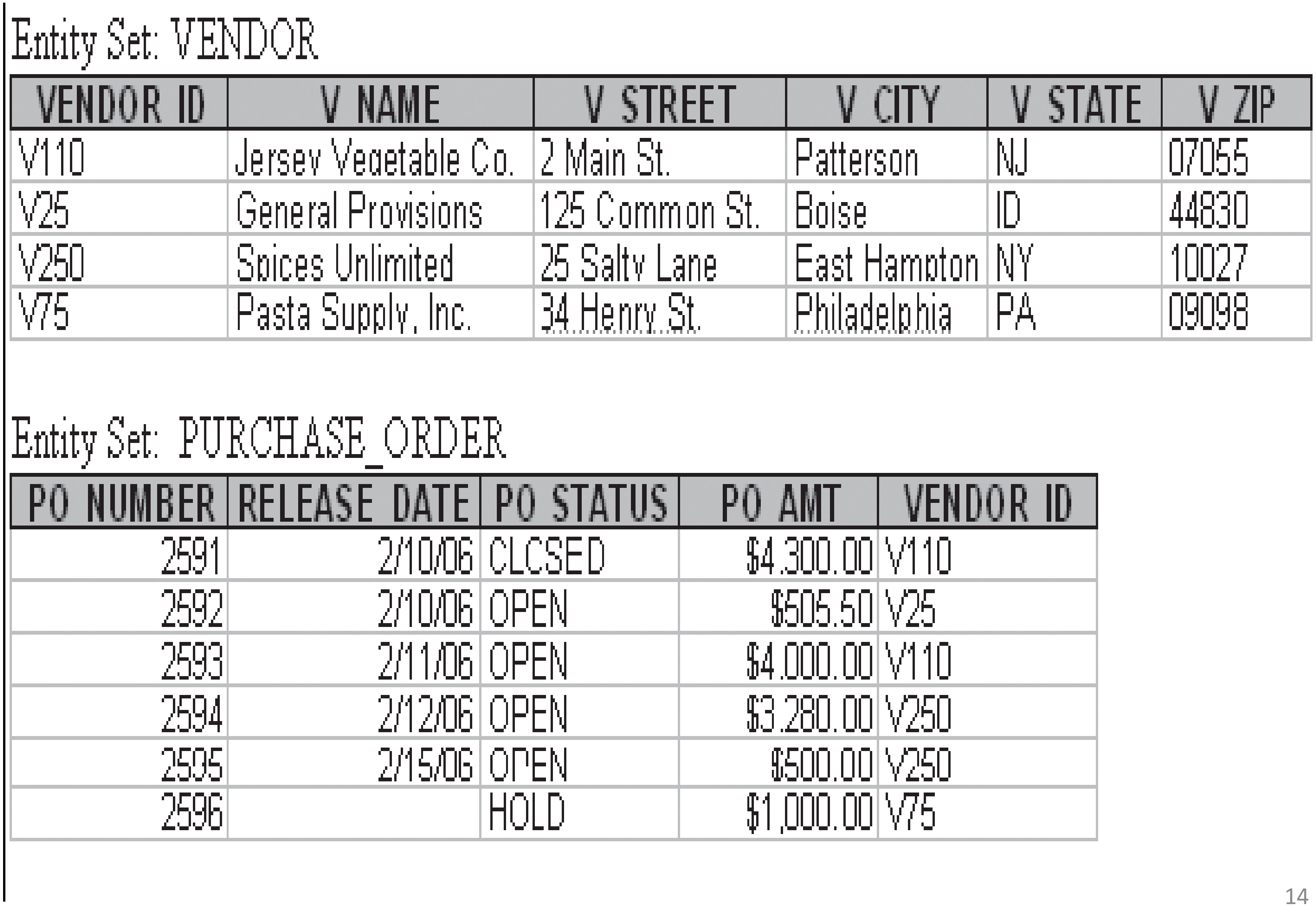
RDB

- Each entity set has unique characteristics, called ***attributes***

- A ***row*** *represents a single entity, or instance of the entity set. A row is sometimes* referred to as a ***record***

- A ***column*** represents the attributes of the entity set. Sometimes columns are referred to as ***fields.***

Example RDB



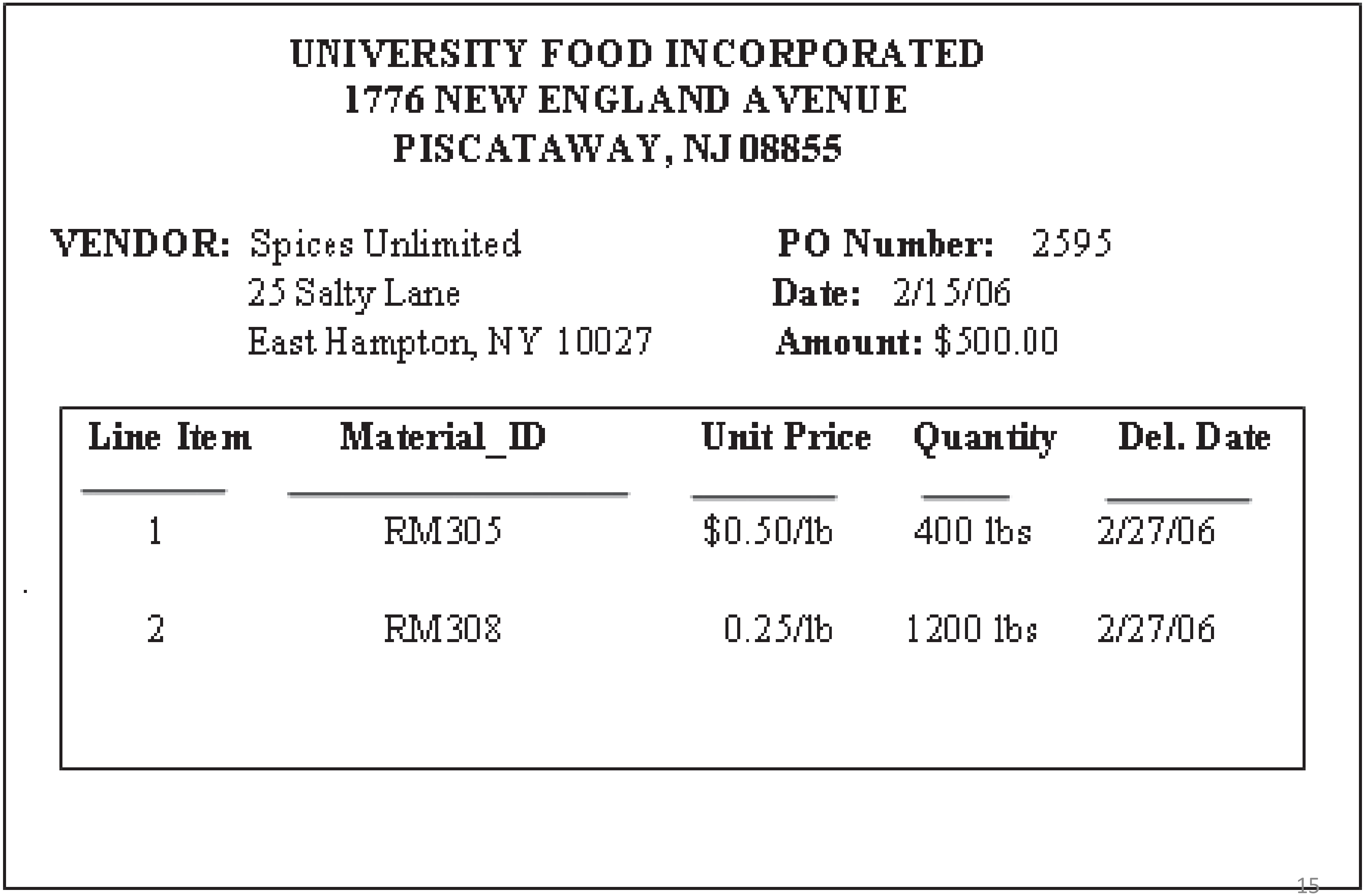
Entity Set: VENDOR

VENDOR ID	V NAME	V STREET	V CITY	V STATE	V ZIP
V110	Jersey Vegetable Co.	2 Main St.	Patterson	NJ	07055
V25	General Provisions	125 Common St.	Boise	ID	44830
V250	Spices Unlimited	25 Salty Lane	East Hampton	NY	10027
V75	Pasta Supply, Inc.	34 Henry St.	Philadelphia	PA	09098

Entity Set: PURCHASE_ORDER

PO NUMBER	RELEASE DATE	PO STATUS	PO AMT	VENDOR ID
2591	2/10/06	CLOSED	$4,300.00	V110
2592	2/10/06	OPEN	$505.50	V25
2593	2/11/06	OPEN	$4,000.00	V110
2594	2/12/06	OPEN	$3,280.00	V250
2595	2/15/06	OPEN	$500.00	V250
2596		HOLD	$1,000.00	V75

Example RDB



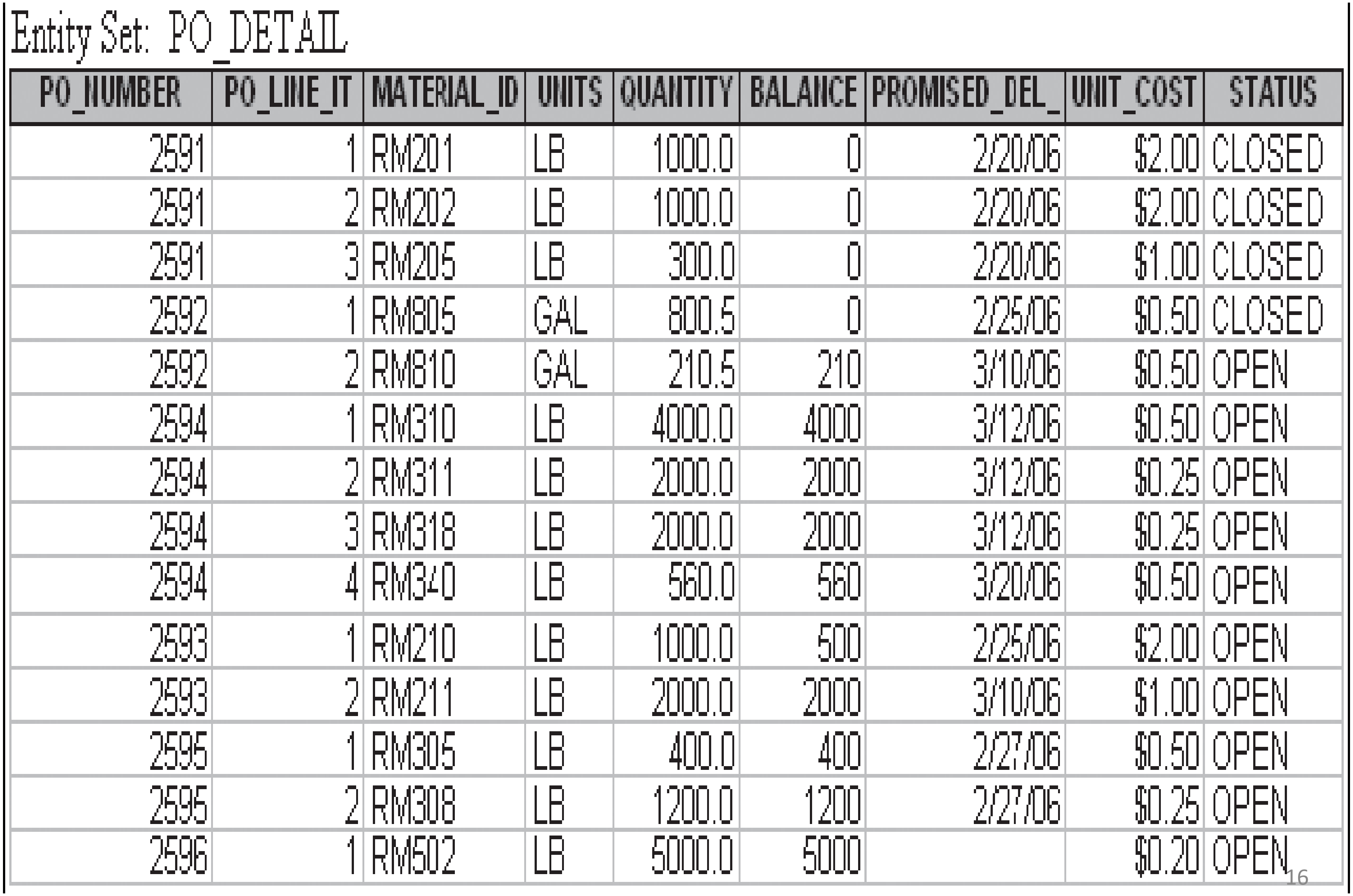
UNIVERSITY FOOD INCORPORATED
1776 NEW ENGLAND AVENUE
PISCATAWAY, NJ 08855

VENDOR: Spices Unlimited
25 Salty Lane
East Hampton, NY 10027

PO Number: 2595
Date: 2/15/06
Amount: $500.00

Line Item	Material_ID	Unit Price	Quantity	Del. Date
1	RM305	$0.50/lb	400 lbs	2/27/06
2	RM308	0.25/lb	1200 lbs	2/27/06

Example RDB



Entity Set: PO_DETAIL

PO_NUMBER	PO_LINE_IT	MATERIAL_ID	UNITS	QUANTITY	BALANCE	PROMISED_DEL_	UNIT_COST	STATUS
2591	1	RM201	LB	1000.0	0	2/20/06	$2.00	CLOSED
2591	2	RM202	LB	1000.0	0	2/20/06	$2.00	CLOSED
2591	3	RM205	LB	300.0	0	2/20/06	$1.00	CLOSED
2592	1	RM805	GAL	800.5	0	2/25/06	$0.50	CLOSED
2592	2	RM810	GAL	210.5	210	3/10/06	$0.50	OPEN
2594	1	RM310	LB	4000.0	4000	3/12/06	$0.50	OPEN
2594	2	RM311	LB	2000.0	2000	3/12/06	$0.25	OPEN
2594	3	RM318	LB	2000.0	2000	3/12/06	$0.25	OPEN
2594	4	RM340	LB	560.0	560	3/20/06	$0.50	OPEN
2593	1	RM210	LB	1000.0	500	2/25/06	$2.00	OPEN
2593	2	RM211	LB	2000.0	2000	3/10/06	$1.00	OPEN
2595	1	RM305	LB	400.0	400	2/27/06	$0.50	OPEN
2595	2	RM308	LB	1200.0	1200	2/27/06	$0.25	OPEN
2596	1	RM502	LB	5000.0	5000		$0.20	OPEN

Key attributes and linking tables

- The two most important key attributes are the ***primary key*** and the ***foreign key*** attributes

- ***Primary key*** attribute is used to uniquely (بشكل منفرد) identify each row (entity)

- For example, in the VENDOR table, the VENDOR_ID is unique. The primary key for the PURCHASE_ORDER table is the PO_NUMBER

Key attributes and linking tables

- PO_NUMBER and PO_LINE_ITEM in PO_DETAIL table represents a unique combination called ***composite primary key attribute***

- The attribute field that relates one or more entities in one table to the primary key attribute in another table is called a ***foreign key.*** In the table PO_DETAIL***,*** the attribute PO_NUMBER serves as a ***foreign key***.

Key attributes and linking tables

- The ***foreign key*** in one table must be related to a ***primary key*** in another table. This is referred to as ***referential integrity السلامة المرجعية***

- When a column is designated as a key attribute (***primary or foreign***), any row of the table must have an entry and not “Null”. Null values may be allowed in non-key attribute columns, but not allowed to exist in key attribute columns.

Data types

- RDBM systems support a variety of data types. Typical data types are *numeric, character or text, date, and currency.*

- Numeric data types are classified as **integer, floating point, or decimal**

- Floating point includes Single precision (4 bytes of data storage) and Double precision (8 bytes of data storage).

Data types

- The **Decimal** data type is a formatted data type in which the DBMS stores the number, including fractional parts, as an integer with up to 12 bytes of data storage.

- **Character or text data** are represented as an alphanumeric string in the range of 1 to 254 characters.

Data types- cont.

- The **DATE** data type tells the DBMS to interpret the field as a date

- The **CURRENCY** data type tells the DBMS that the numerical value is a monetary value.

Structured query language (SQL)

SQL

- The relational database community has defined a standard language for manipulating data in a database called ***Structured Query Language (SQL).***

- The American National Standards Institute (ANSI) has standardized SQL

- SQL is a nonprocedural language

- In a nonprocedural language, you are not concerned with the details of how the work gets done, you only have to define what you want to have done.

SQL

- There are about 30 standard instructions in the basic SQL command set.

- The standard SQL instructions allow the user to perform operations for the following purposes:

  (1) to create a database and its table structure,

  (2) to manage the data in the database tables, and

  (3) to summarize the data into useful information for decision making.

SQL: Creating the database and table structure

- The CREATE command keyword is used for creating databases and tables

CREATE DATABASE <database name>;

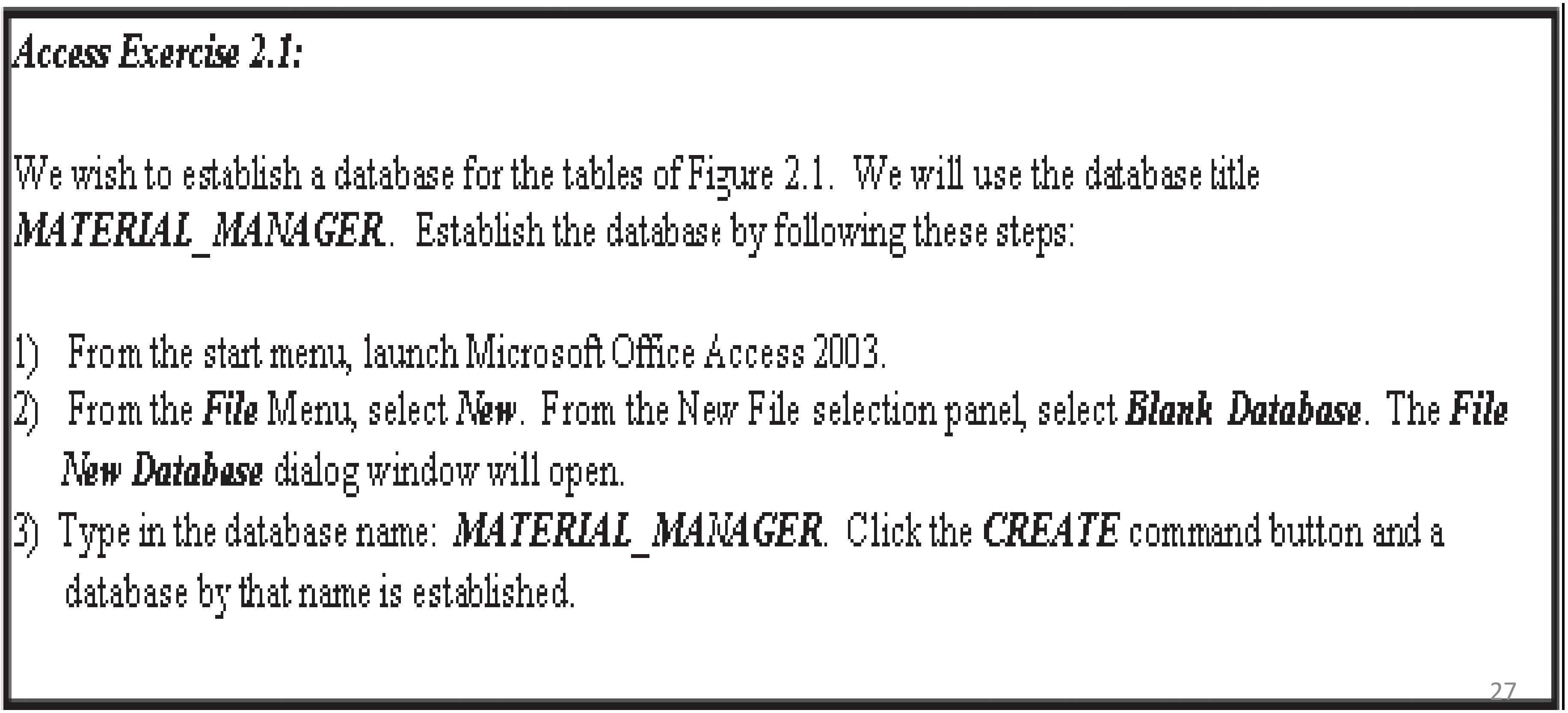
When this command is executed, a database with the given name is created.

- The Access RDBMS does not support the CREATE DATABASE command.

- Instead, it provides a Windows menu-driven route to establishing the database.

Creating database in Access

- Create the database in figure 2.1 in the book , page 29 (VENDOR, PURCHASE_ORDER, PO_DETAIL)



Access Exercise 2.1:

We wish to establish a database for the tables of Figure 2.1. We will use the database title ***MATERIAL_MANAGER***. Establish the database by following these steps:

1) From the start menu, launch Microsoft Office Access 2003.
2) From the ***File*** Menu, select ***New***. From the New File selection panel, select ***Blank Database***. The ***File New Database*** dialog window will open.
3) Type in the database name: ***MATERIAL_MANAGER***. Click the ***CREATE*** command button and a database by that name is established.

Writing SQL commands

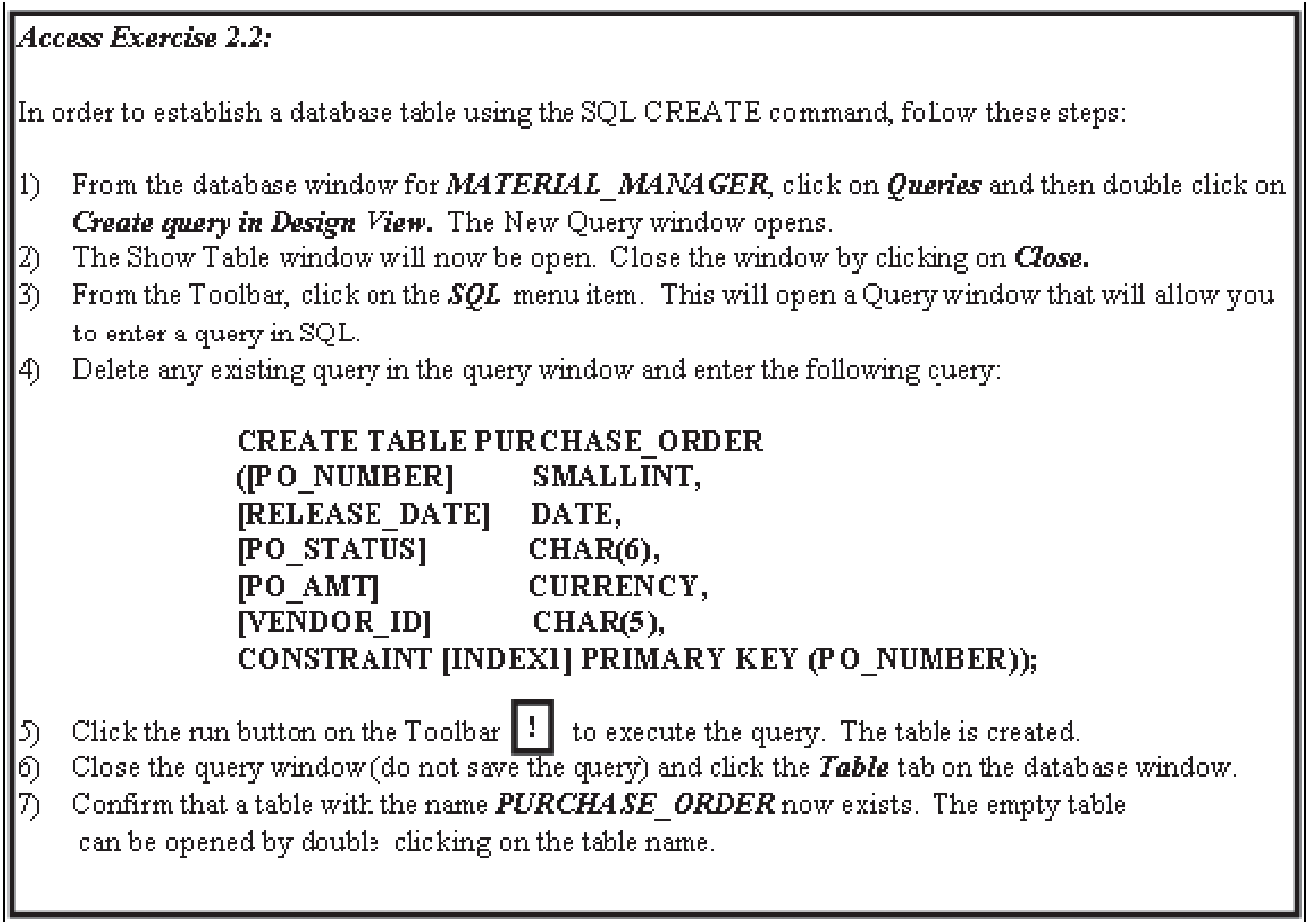
The following conventions will be used to describe SQL commands:

- The command will be written in CAPITAL LETTERS Commands are usually followed by arguments
- Arguments are placed within angle brackets, < >. If an argument is optional, it will be bounded by square brackets, [< >].

Writing SQL commands

- The syntax for creating a table within a database uses the CREATE keyword as follows:

```
CREATE TABLE <table name>
(<[attribute1 name]> <data type>,
<[attribute2 name]> <data type>,
… … …
<[attributeN name]> <data type>);
```



Access Exercise 2.2:

In order to establish a database table using the SQL CREATE command, folow these steps:

1) From the database window for ***MATERIAL_MANAGER***, click on ***Queries*** and then double click on ***Create query in Design View.*** The New Query window opens.
2) The Show Table window will now be open. Close the window by clicking on ***Close.***
3) From the Toolbar, click on the ***SQL*** menu item. This will open a Query window that will allow you to enter a query in SQL.
4) Delete any existing query in the query window and enter the following query:

```
CREATE TABLE PURCHASE_ORDER
([PO_NUMBER]        SMALLINT,
[RELEASE_DATE]      DATE,
[PO_STATUS]         CHAR(6),
[PO_AMT]            CURRENCY,
[VENDOR_ID]         CHAR(5),
CONSTRAINT [INDEX1] PRIMARY KEY (PO_NUMBER));
```

5) Click the run button on the Toolbar ! to execute the query. The table is created.
6) Close the query window (do not save the query) and click the ***Table*** tab on the database window.
7) Confirm that a table witk the name ***PURCHASE_ORDER*** now exists. The empty table can be opened by doubl: clicking on the table name.

Writing SQL commands

- By limiting the length of the character strings for PO_STATUS and VENDOR_ID, computer memory storage space is saved

- If we did not limit the character length, the DBMS would have defaulted the field to 50 characters. The largest text field size is 255 characters.

Writing SQL commands

- In Access Exercise 2.2, the constraint clause is used to constrain the primary key to PO_NUMBER. The attribute PO_NUMBER is constrained to be **Not Null and Unique.**

- Within the constraint clause, an index object has also been defined. it speeds up the processing times because Access uses the index to process records in a defined order.

Writing SQL commands

- Index allows the DBMS search algorithm to locate a particular record in a table faster than would be possible without the index.

- This can be helpful in databases containing many large tables.

Writing SQL commands

```
CREATE TABLE VENDOR
([VENDOR_ID] CHAR(5),
[V_NAME] CHAR(20),
[V_STREET] CHAR(20),
[V_CITY] CHAR(20),
[V_STATE] CHAR(2),
[V_ZIP] CHAR(5),
CONSTRAINT [INDEX2] PRIMARY KEY
   (VENDOR_ID));
```

Writing SQL commands

```
CREATE TABLE PO_DETAIL
([PO_NUMBER] SMALLINT,
[PO_LINE_ITEM] SMALLINT,
[MATERIAL_ID] CHAR(10),
[UNITS] CHAR(4),
[QUANTITY] SINGLE,
[BALANCE] SINGLE,
[PROMISED_DEL_DATE] DATE,
[UNIT_COST] CURRENCY,
[STATUS] CHAR(6),
CONSTRAINT [INDEX3]
PRIMARY KEY (PO_NUMBER, PO_LINE_ITEM));
```

SQL: Managing the data in the database table

- There are several keyword commands for populating and manipulating data in the database: the keywords **INSERT, SELECT, UPDATE, and DELETE**

1. INSERT Keyword**:**

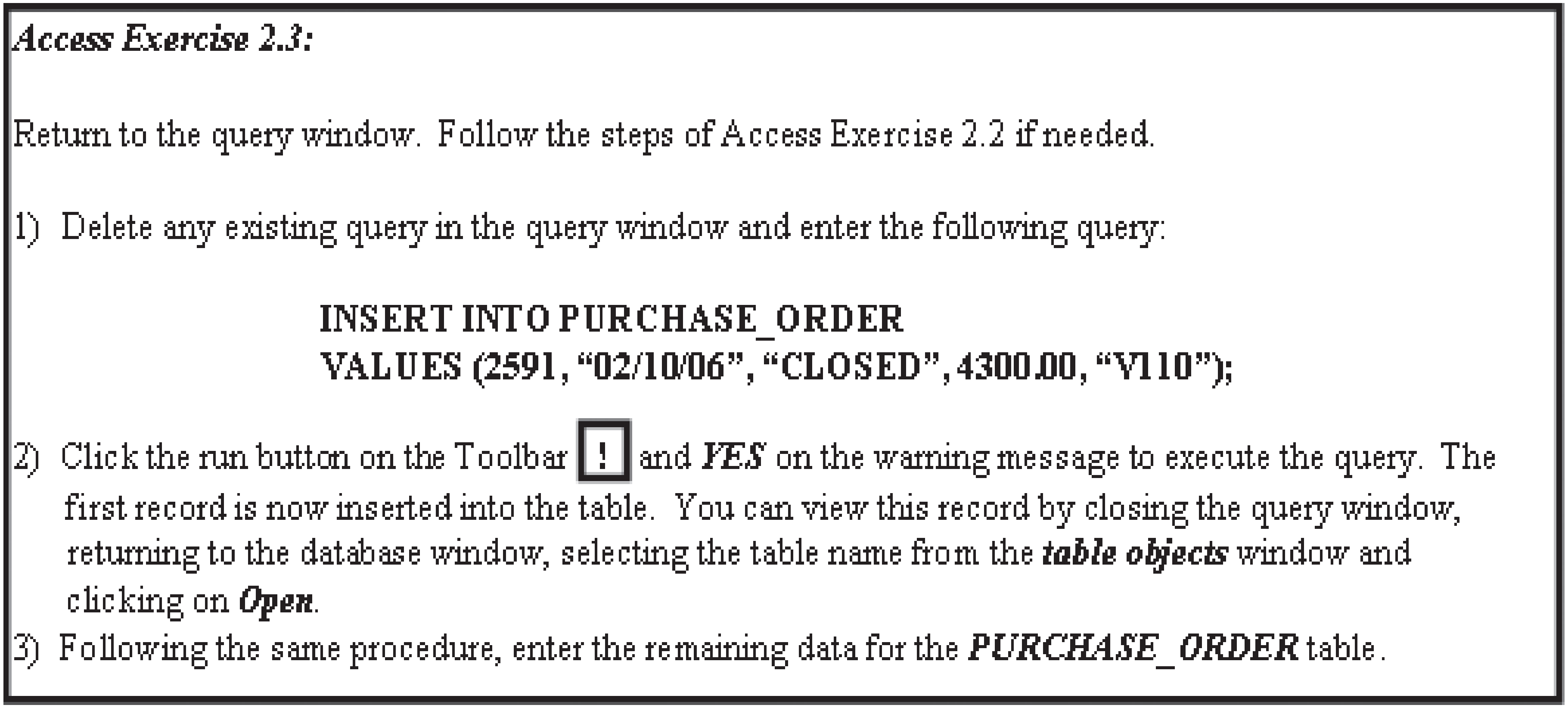
- Records can be placed into empty tables using the SQL **INSERT command.** The syntax is as follows:

```
INSERT INTO <table name>
([<attribute1 name>], <[attribute2 name]>,...)
VALUES (<value1>, <value2>,...);
```

Examples insertions

INSERT INTO VENDOR

VALUES (“V110”, “Jersey Vegetables Co.”, “2 Main St.” , “Patterson”, “NJ”, “07055”);



Access Exercise 2.3:

Return to the query window. Follow the steps of Access Exercise 2.2 if needed.

1) Delete any existing query in the query window and enter the following query:

```
INSERT INTO PURCHASE_ORDER
VALUES (2591, “02/10/06”, “CLOSED”, 4300.00, “V110”);
```

2) Click the run button on the Toolbar ! and ***YES*** on the warning message to execute the query. The first record is now inserted into the table. You can view this record by closing the query window, returning to the database window, selecting the table name from the ***table objects*** window and clicking on ***Open***.

3) Following the same procedure, enter the remaining data for the ***PURCHASE_ORDER*** table.

Direct insertion

- In addition to the use of SQL, instead of using SQL and the INSERT keyword, the user can open a table and directly insert data into the rows and columns of the table.

SQL: Managing the data in the database table

2. SELECT Keyword

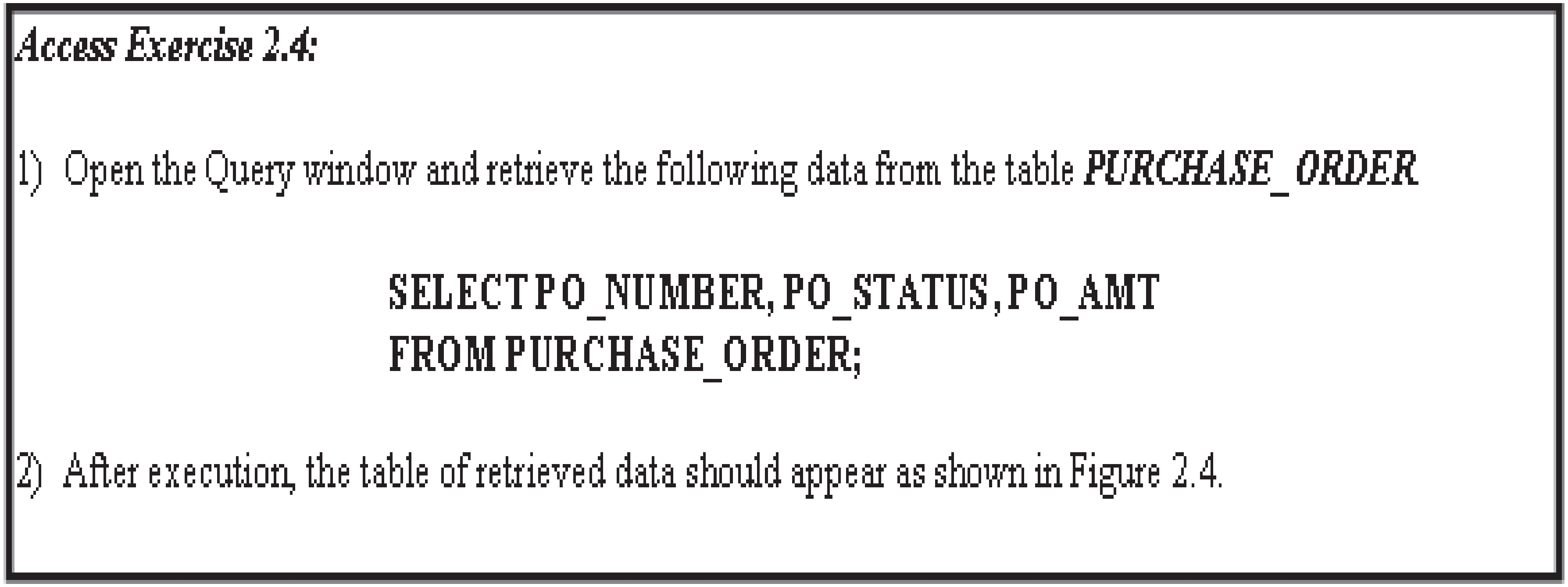
The most common command used in retrieving and manipulating data in a table is the **SELECT** Keyword

```
SELECT [DISTINCT] <attributes/*>
FROM <table name>
WHERE <condition>
ORDER BY <attribute name> ASC/DESC;
```

```
SELECT *
FROM PURCHASE_ORDER;
```

The wildcard *** identifies all attribute** fields of the table.

2. SELECT Keyword – Cont.

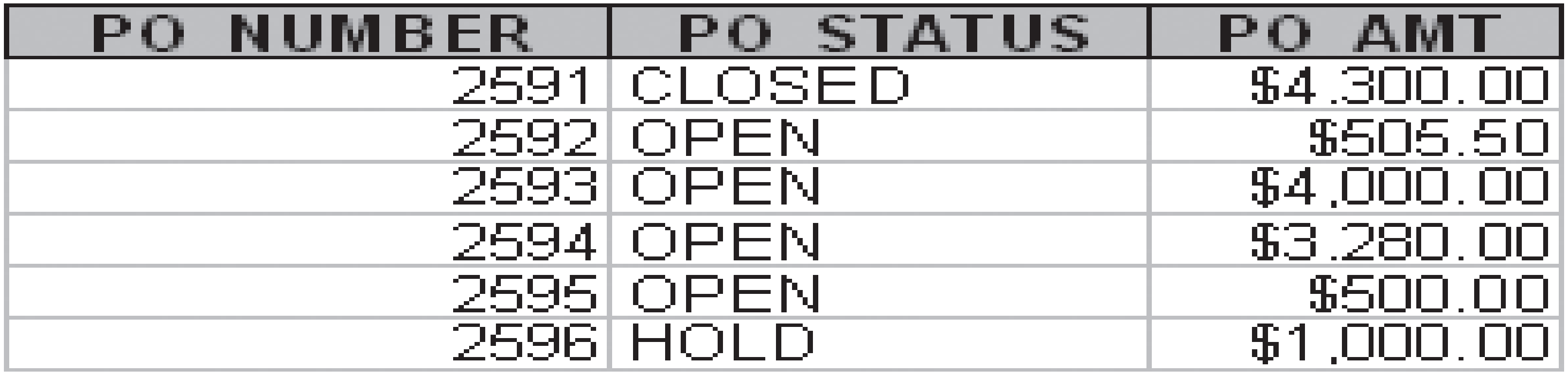


Access Exercise 2.4:

1) Open the Query window and retrieve the following data from the table ***PURCHASE_ORDER***

```
SELECT PO_NUMBER, PO_STATUS, PO_AMT
FROM PURCHASE_ORDER;
```

2) After execution, the table of retrieved data should appear as shown in Figure 2.4.



PO NUMBER	PO STATUS	PO AMT
2591	CLOSED	$4.300.00
2592	OPEN	$505.50
2593	OPEN	$4,000.00
2594	OPEN	$3.280.00
2595	OPEN	$500.00
2596	HOLD	$1,000.00

2. SELECT Keyword – Cont.

- The record of retrieved data in exercise 2.4 is called a “recordset.”

- A ***recordset*** is a view of the data from one or more tables, selected and sorted as specified by the query

- This can be done by using the **ORDER BY** clause, followed by the attribute name of the column on which you wish to impose the order.

2. SELECT Keyword – Cont.

- If the column is a numeric, date, or currency data type, the order will be determined by the magnitude of the number. The default is to use ascending order

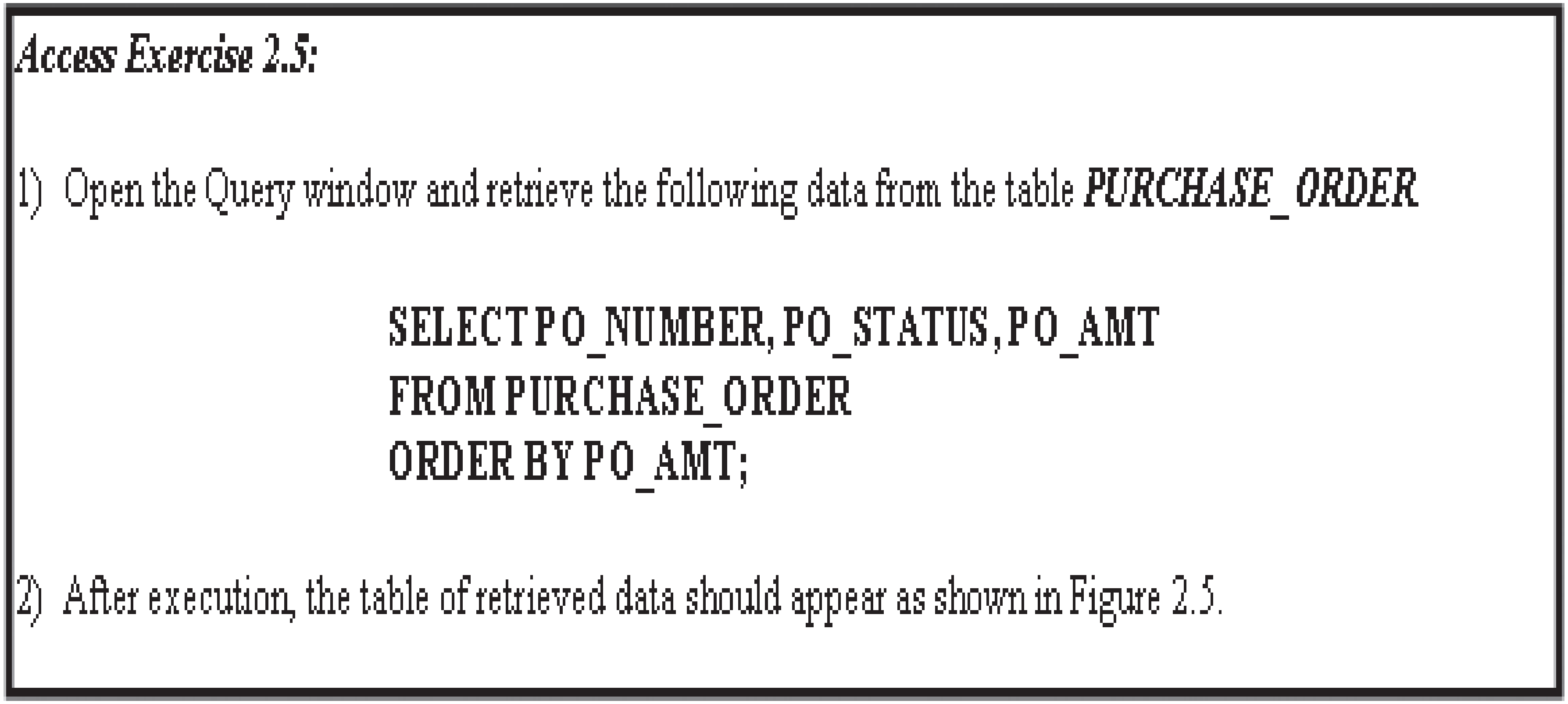
- For descending order, the keyword **DESC** is used following the attribute name

- If the column is a text data type, the order is determined by ASCII equivalent.

- The computer stores text data by its numeric equivalent in ASCII. Thus, the letter *A is equivalent to the decimal value 65 in ASCII, B is 66, C is 67, and so* forth. The lower case letter *a has a value of 97, b is 98, c is 99, and so forth*

- *Therefore, in sorting* alphabetic strings is ascending order, *A precedes B, and B precedes a, which precedes b, and* so on.

2. SELECT Keyword – Cont.

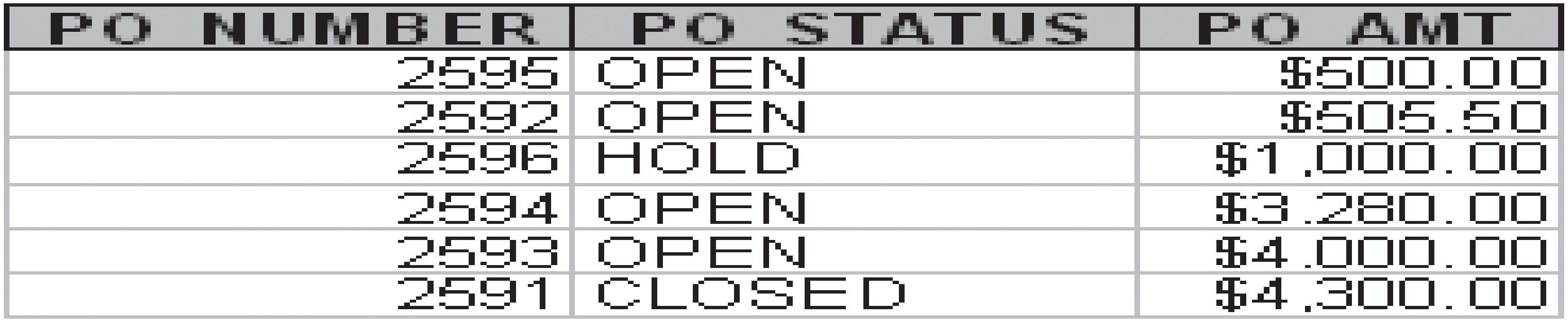


Access Exercise 2.5:

1) Open the Query window and retrieve the following data from the table ***PURCHASE_ORDER***

```
SELECT PO_NUMBER, PO_STATUS, PO_AMT
FROM PURCHASE_ORDER
ORDER BY PO_AMT;
```

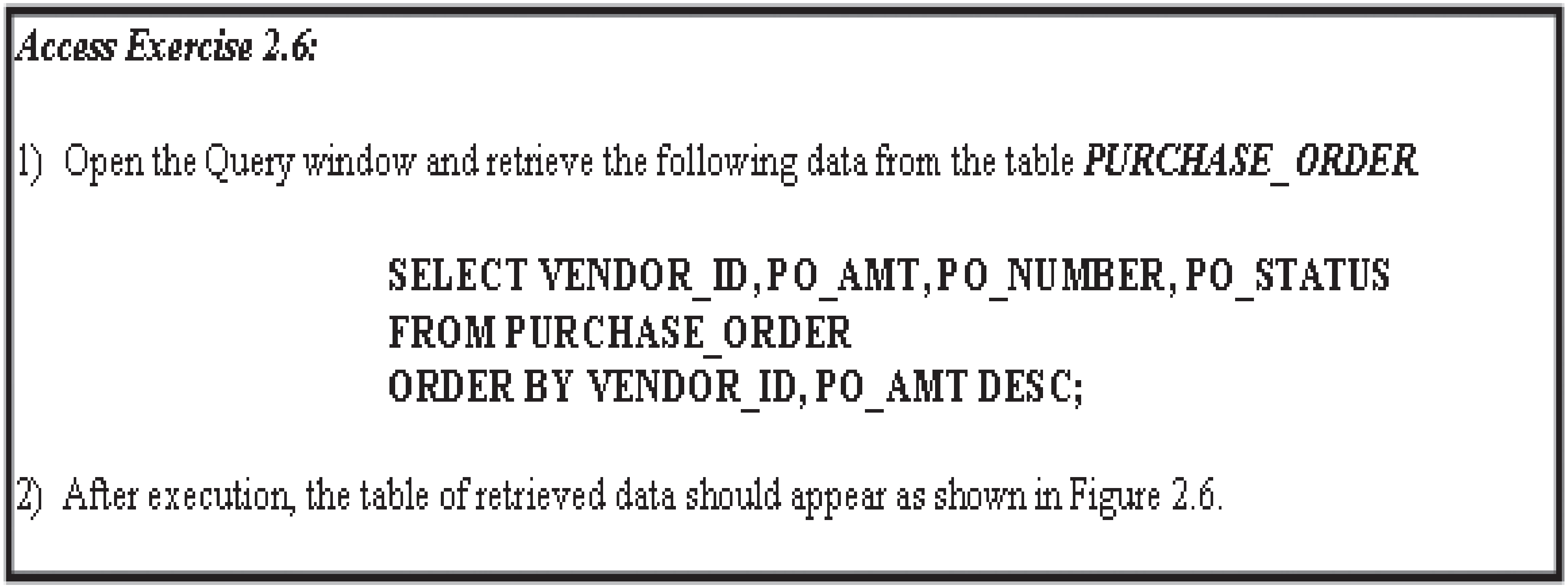
2) After execution, the table of retrieved data should appear as shown in Figure 2.5.



PO NUMBER	PO STATUS	PO AMT
2595	OPEN	$500.00
2592	OPEN	$505.50
2596	HOLD	$1,000.00
2594	OPEN	$3,280.00
2593	OPEN	$4,000.00
2591	CLOSED	$4,300.00

2. SELECT Keyword – Cont.

- Sometimes it is desirable to have an ordering within an ordering. For example, it may be desirable to order purchase orders by VENDOR_ID and, within each vendor group, to order the PO_AMT.



Access Exercise 2.6:

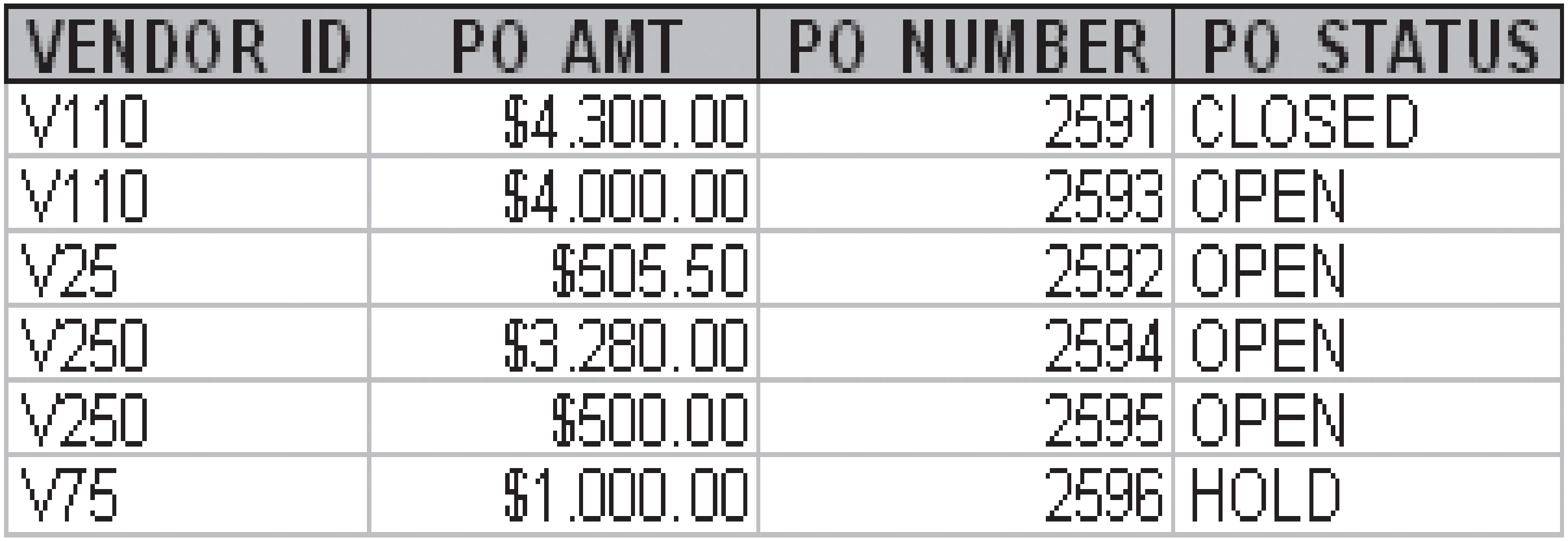
1) Open the Query window and retrieve the following data from the table ***PURCHASE_ORDER***

```
SELECT VENDOR_ID,PO_AMT,PO_NUMBER, PO_STATUS
FROM PURCHASE_ORDER
ORDER BY VENDOR_ID, PO_AMT DESC;
```

2) After execution, the table of retrieved data should appear as shown in Figure 2.6.

2. SELECT Keyword – Cont.

- The command returns:



VENDOR ID	PO AMT	PO NUMBER	PO STATUS
V110	$4.300.00	2591	CLOSED
V110	$4.000.00	2593	OPEN
V25	$505.50	2592	OPEN
V250	$3.280.00	2594	OPEN
V250	$500.00	2595	OPEN
V75	$1.000.00	2596	HOLD

2. SELECT Keyword – Cont.

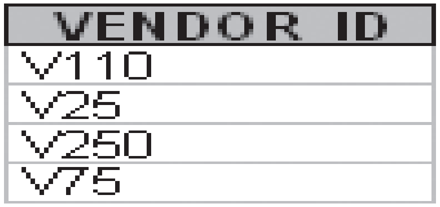
- The **DISTINCT keyword** in the **SELECT** clause allows the user to sort a single column of a table and to return a list of the unique entries in that column

- For example, the following command will return the unique VENDOR_IDs in the PURCHASE_ORDERS table:

```
SELECT DISTINCT VENDOR_ID
FROM PURCHASE_ORDER;
```

2. SELECT Keyword – Cont.

- The command returns:



VENDOR ID
V110
V25
V250
V75

- It is often of interest to select out specific rows from a table based on a criterion. This is done using the **WHERE** clause of the **SELECT** command:

SELECT *

FROM PURCHASE_ORDER

WHERE VENDOR_ID="V110";

2. SELECT Keyword – Cont.

- Note the use of the equal (=) sign to indicate the row selection criteria. The use of comparison operators is a common method of indicating the selection criteria. The comparison operators are as follows:

= equal to

!= not equal to

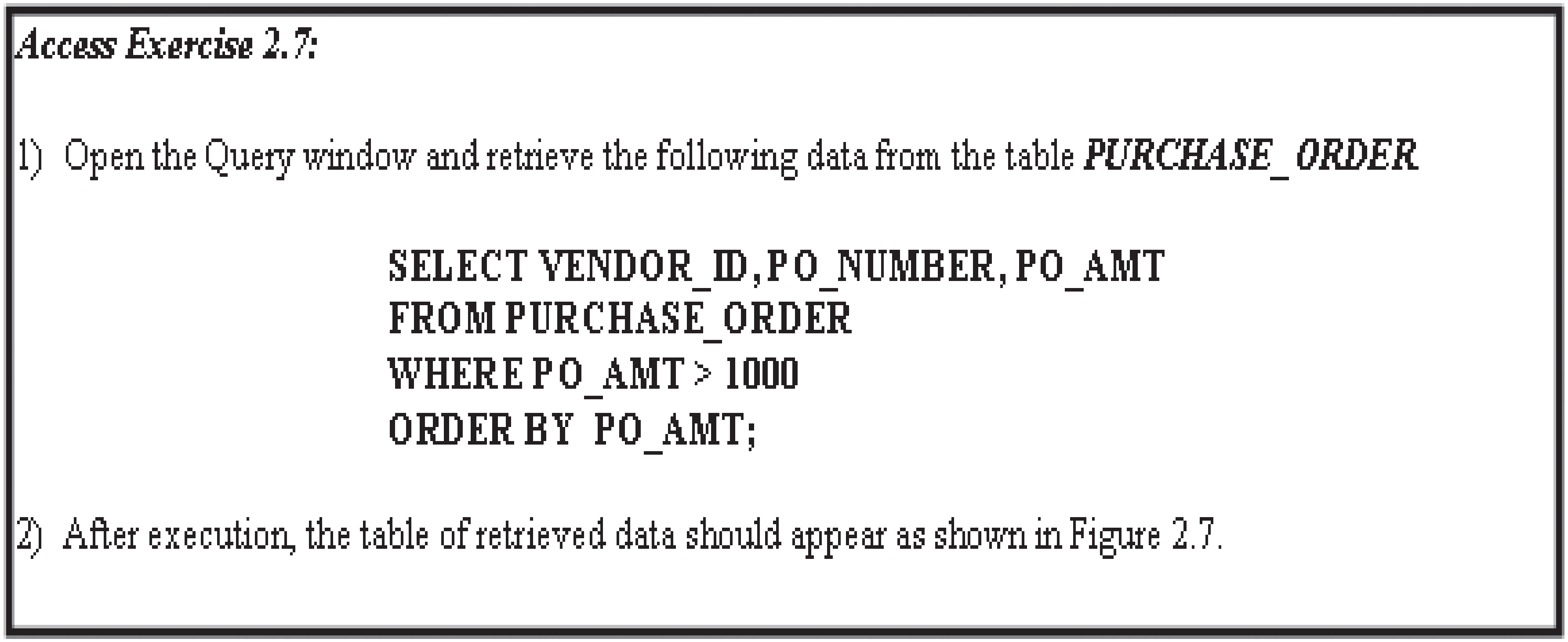
< less than

<= less than or equal to

> greater than

>= greater than or equal to

2. SELECT Keyword – Cont.

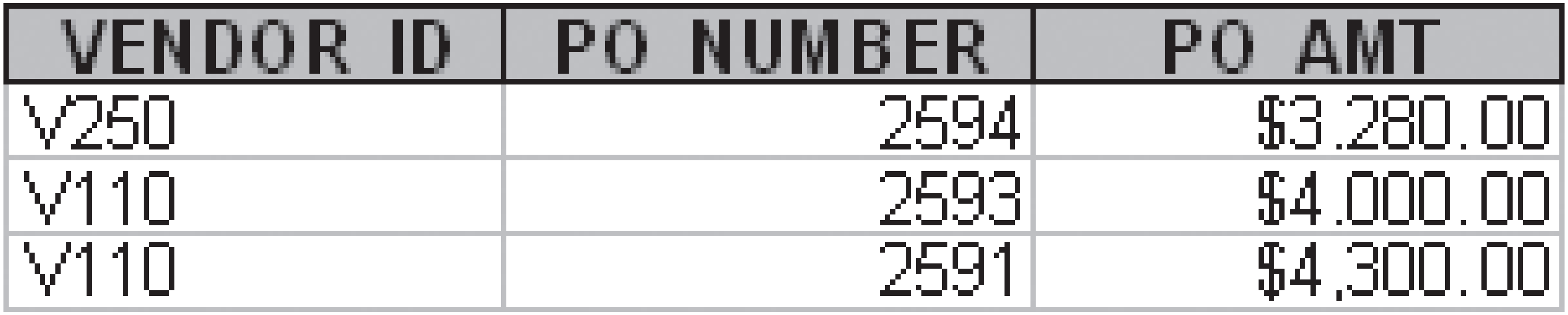


Access Exercise 2.7:

1) Open the Query window and retrieve the following data from the table ***PURCHASE_ORDER***

```
SELECT VENDOR_ID, PO_NUMBER, PO_AMT
FROM PURCHASE_ORDER
WHERE PO_AMT > 1000
ORDER BY  PO_AMT;
```

2) After execution, the table of retrieved data should appear as shown in Figure 2.7.



VENDOR_ID	PO_NUMBER	PO_AMT
V250	2594	$3,280.00
V110	2593	$4,000.00
V110	2591	$4,300.00

2. SELECT Keyword – Cont.

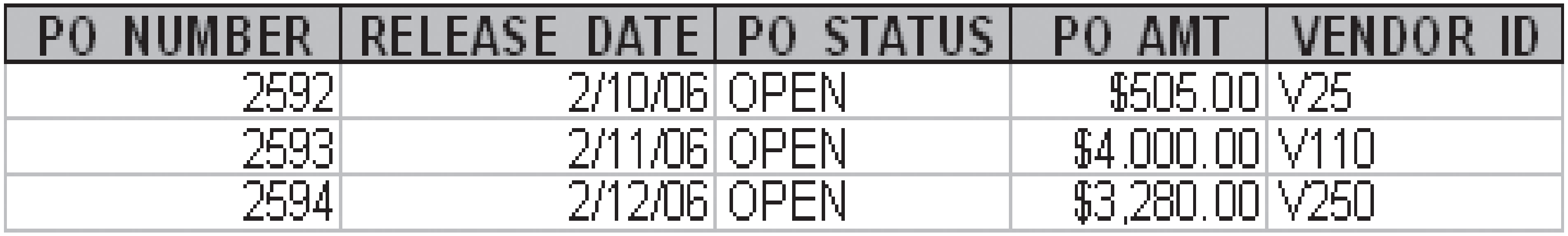
- There may be more than one selection criterion for a retrieval. This can be handled by extending the WHERE clause using logical operators. The ***logical operators are AND, OR, and NOT.***

- For example, suppose we wish to look at all open orders that are above $500. This would require the logical AND, since we want the orders that are open AND greater than $500. The WHERE clause is extended using logical operators as follows:

```
SELECT *
FROM PURCHASE_ORDER
WHERE PO_STATUS=“OPEN”
AND PO_AMT>500;
```

2. SELECT Keyword – Cont.

- The command returns:



PO NUMBER	RELEASE DATE	PO STATUS	PO AMT	VENDOR ID
2592	2/10/06	OPEN	$505.00	V25
2593	2/11/06	OPEN	$4,000.00	V110
2594	2/12/06	OPEN	$3,280.00	V250

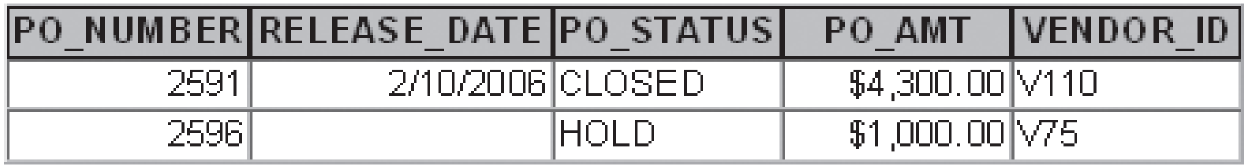
- Access does not support != (not equal to). Instead, the NOT operator is used to return the complementary set of an operation. For example:

SELECT *

FROM PURCHASE_ORDER

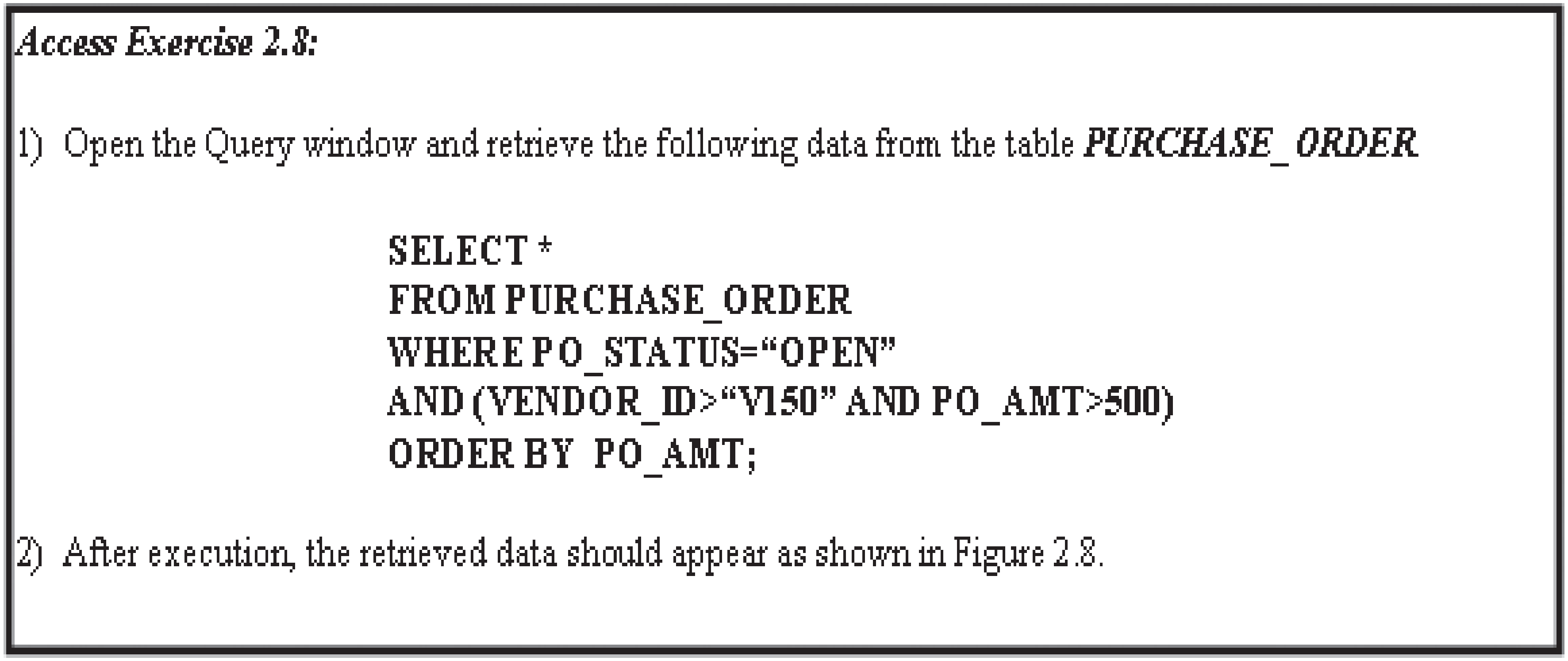
WHERE NOT PO_STATUS = “OPEN”;

- This will return all purchase orders that are not open as follows:



PO_NUMBER	RELEASE_DATE	PO_STATUS	PO_AMT	VENDOR_ID
2591	2/10/2006	CLOSED	$4,300.00	V110
2596		HOLD	$1,000.00	V75

2. SELECT Keyword – Cont.

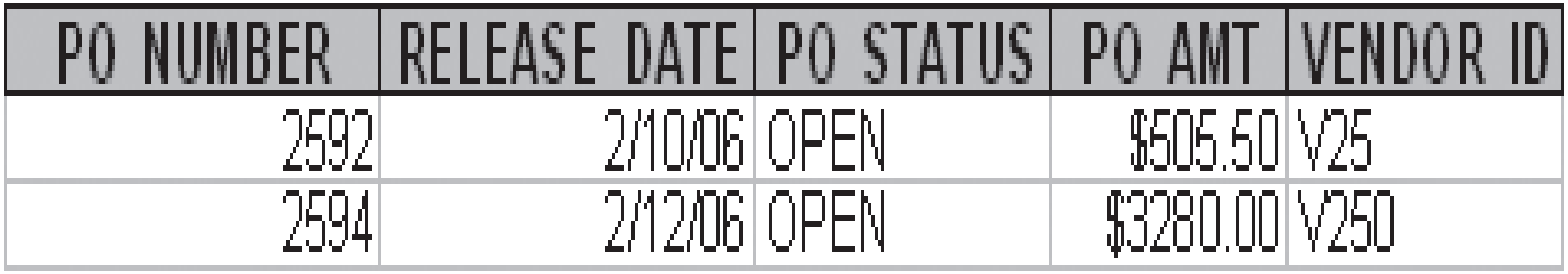


Access Exercise 2.8:

1) Open the Query window and retrieve the following data from the table ***PURCHASE_ORDER***

```
SELECT *
FROM PURCHASE_ORDER
WHERE PO_STATUS=“OPEN”
AND (VENDOR_ID>“V150” AND PO_AMT>500)
ORDER BY  PO_AMT;
```

2) After execution, the retrieved data should appear as shown in Figure 2.8.



PO NUMBER	RELEASE DATE	PO STATUS	PO AMT	VENDOR ID
2592	2/10/06	OPEN	$505.50	V25
2594	2/12/06	OPEN	$3280.00	V250

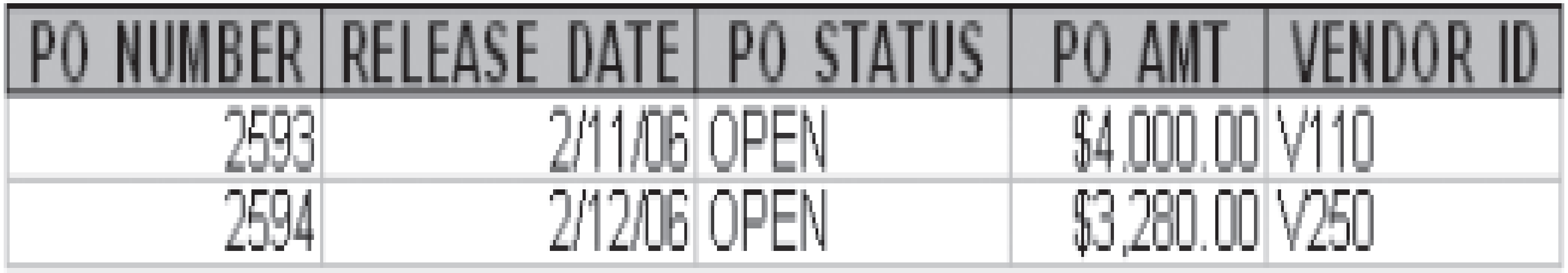
2. SELECT Keyword – Cont.

- Another useful operator is the **BETWEEN** keyword. This allows the user to specify a range of values over which the data will be retrieved:

```
SELECT *
FROM PURCHASE_ORDER
WHERE ((PO_AMT BETWEEN 505 AND 4000)
AND (RELEASE_DATE BETWEEN #2/11/06# AND
   #2/13/06#));
```

2. SELECT Keyword – Cont.

This query returns the following result:



PO NUMBER	RELEASE DATE	PO STATUS	PO AMT	VENDOR ID
2593	2/11/06	OPEN	$4,000.00	V110
2594	2/12/06	OPEN	$3,280.00	V250

- Also, the complement of BETWEEN is NOT BETWEEN, which will return the complement set from the table. Also note the use of the pound sign (#) around the DATE data type values.
- In Access SELECT command statements, the # sign is used to indicate a DATE data type.

2. SELECT Keyword – Cont.

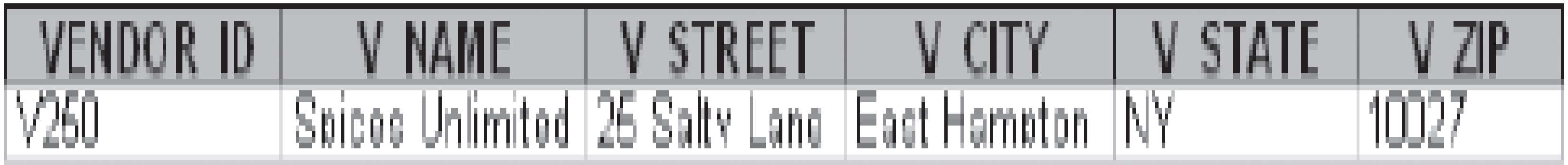
- SQL also provides a predicate operator that allows a search to be done on strings that are a partial match to the predicate. This is done using the **LIKE keyword:**

SELECT *

FROM VENDOR

WHERE V_NAME LIKE "SPICE*";

This query returns the following:



VENDOR ID	V NAME	V STREET	V CITY	V STATE	V ZIP
V250	Spices Unlimited	25 Salty Lane	East Hampton	NY	10027

3. UPDATE Keyword

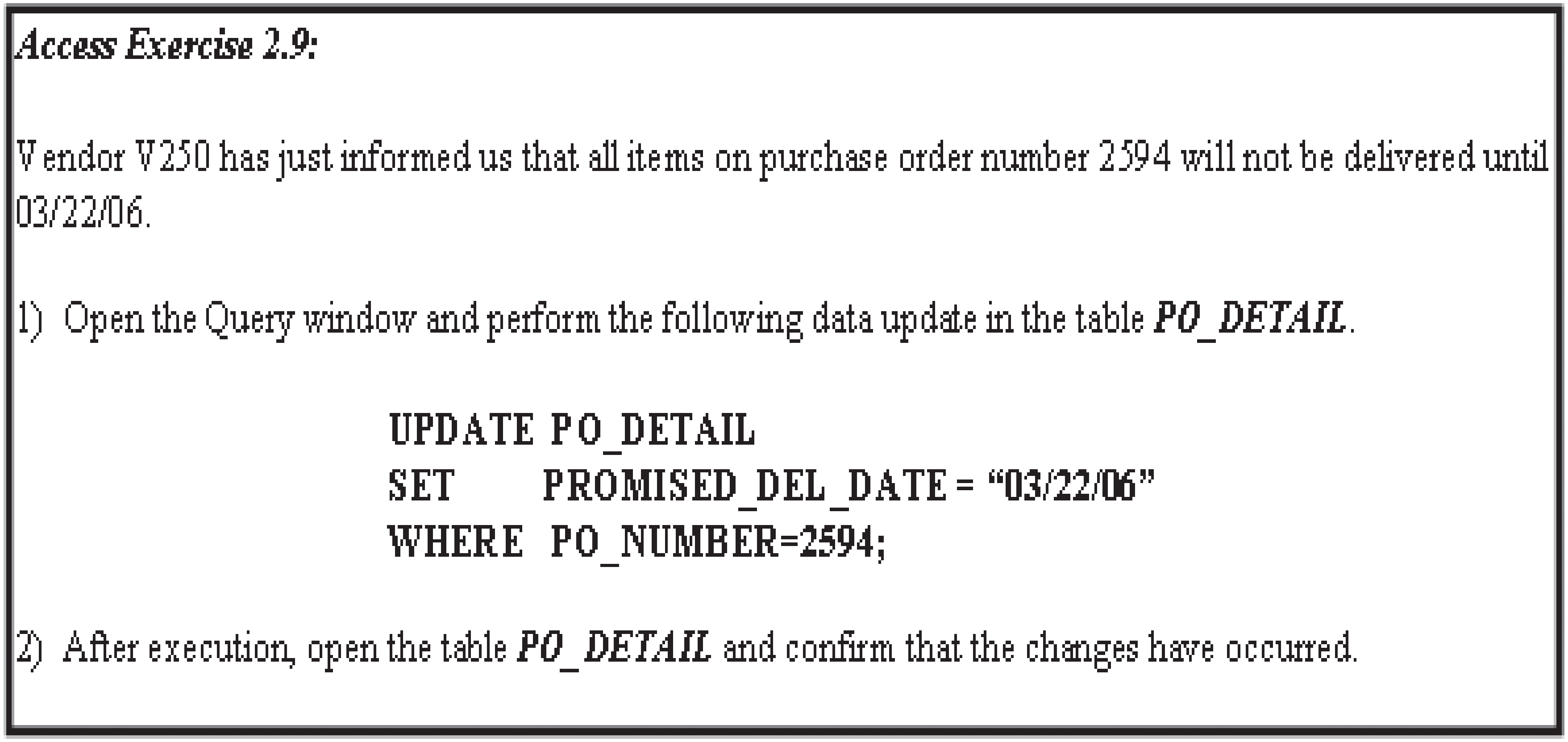
- The **UPDATE** keyword allows the user to replace existing values in a table with new values

- The syntax of the UPDATE command is as follows:

```
UPDATE <table name>
SET <attribute name> = <value/expression> [...]
[WHERE <condition>];
```

3. UPDATE Keyword

- So, for example, if the attribute is the price of a product and the company raises all its prices by 2%, the SET clause could read as follows:



Access Exercise 2.9:

Vendor V250 has just informed us that all items on purchase order number 2594 will not be delivered until 03/22/06.

1) Open the Query window and perform the following data update in the table ***PO_DETAIL***.

```
UPDATE PO_DETAIL
SET     PROMISED_DEL_DATE = "03/22/06"
WHERE  PO_NUMBER=2594;
```

2) After execution, open the table ***PO_DETAIL*** and confirm that the changes have occurred.

4. DELETE Keyword

- The **DELETE** keyword allows the user to remove one or more rows from a table (i.e., delete one or more records).

- The syntax for the DELETE command is as follows:

```
DELETE FROM <table name>
WHERE <condition>;

DELETE FROM PO_DETAIL
WHERE PO_NUMBER = 2596;
DELETE FROM PURCHASE_ORDER
WHERE PO_NUMBER = 2596;
```

SQL: Converting data into information

- Converting the data into useful information for decision making involves making computations (processing) on the fields within the tables

- Some basic arithmetic and logical functions are built into SQL that enable relatively simple calculations

- In addition, compiling information for summary purposes often involves retrieving information from more than one table at the same time.

Aggregate Functions in SQL

- The aggregate functions allow the user to specify a summary mathematical operation with a keyword

- The basic aggregate functions of the SQL are **AVG, SUM, MIN, MAX, and COUNT**

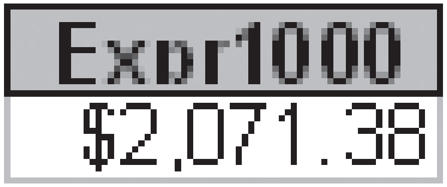
- Syntax of the aggregate functions is as follows:

```
SELECT AGGREGATE FUNCTION ([DISTINCT] <attribute name>)
FROM <table name>
WHERE <condition>
GROUP BY <attribute name> [HAVING <condition>];
```

Aggregate Functions in SQL – Cont.

- The following examples illustrate the use of the aggregate functions:

```
SELECT AVG(PO_AMT)
FROM PURCHASE_ORDER
WHERE PO_STATUS="OPEN";
```

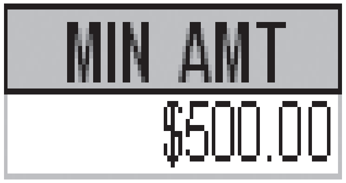


Expr1000
$2,071.38

Aggregate Functions in SQL – Cont.

- Expr1000 is a default sequential number assigned by Access to and refers to the expression AVG(PO_AMT)
- However, a label can be specified by the programmer using the AS clause. Consider the use of the **AS** clause in the following example. The command

```
SELECT MIN(PO_AMT) AS MIN_AMT
FROM PURCHASE_ORDER;
```

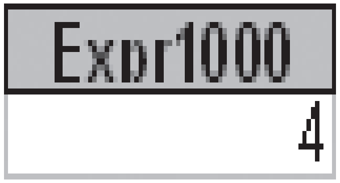


MIN AMT
$500.00

Aggregate Functions in SQL – Cont.

- The **COUNT** keyword is used to return a count of the number of rows in a column having entries that satisfy the **WHERE** clause of the command:

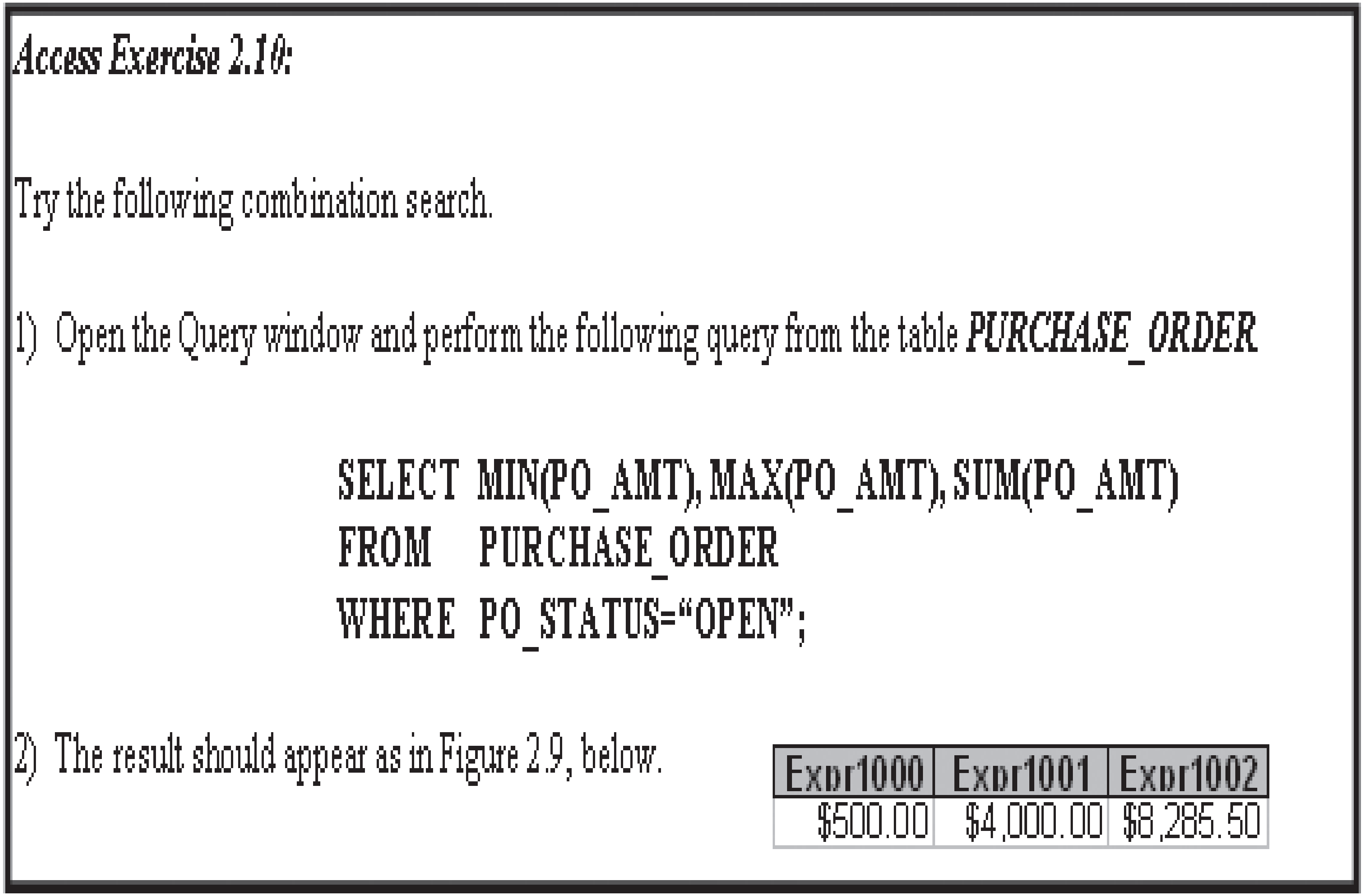
```
SELECT COUNT(PO_STATUS)
FROM PURCHASE_ORDER
WHERE PO_STATUS=“OPEN”;
```



Expr1000
4

- This command returns a count of the number of purchase orders in the database that are still open

Aggregate Functions in SQL – Cont.



Access Exercise 2.10:

Try the following combination search.

1) Open the Query window and perform the following query from the table ***PURCHASE_ORDER***

```
SELECT MIN(PO_AMT), MAX(PO_AMT), SUM(PO_AMT)
FROM   PURCHASE_ORDER
WHERE  PO_STATUS="OPEN";
```

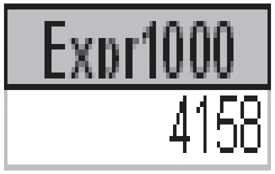
2) The result should appear as in Figure 2.9, below.

Expr1000	Expr1001	Expr1002
$500.00	$4,000.00	$8,285.50

Aggregate Functions in SQL – Cont.

- It is also possible to embed an arithmetic operation in an aggregate function SELECT clause or to use a logical operator for multiple criteria in the WHERE clause of an aggregate function:

```
SELECT SUM(PO_AMT)*1.10
FROM PURCHASE_ORDER
WHERE VENDOR_ID=“V250”
AND (PO_NUMBER=2594 OR PO_NUMBER=2595);
```



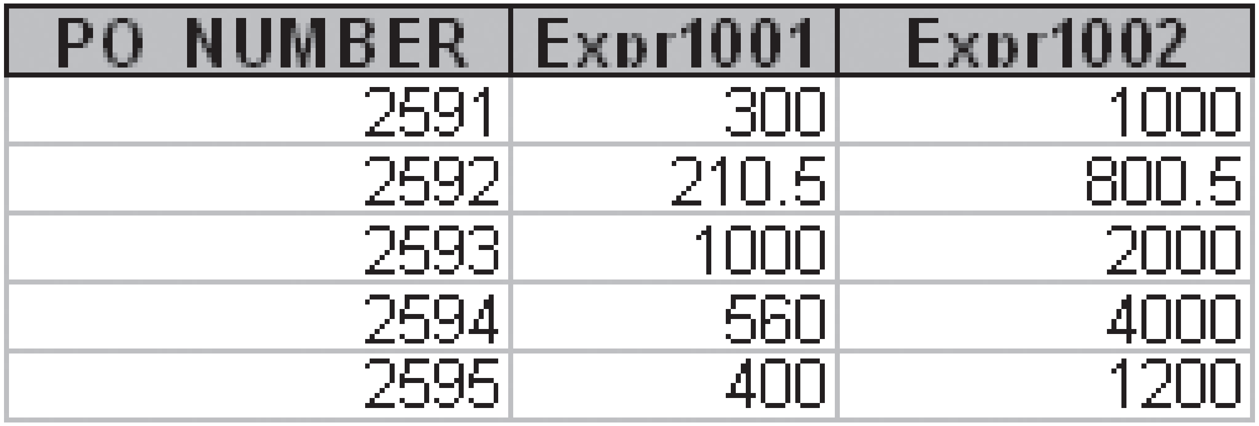
Expr1000
4158

Grouping Data

- The **GROUP BY** clause can be used to group data from an aggregate function by a column attribute
- This allows you to display the results of several aggregates by some meaningful summary
- For example, suppose you want to summarize aggregates of data from the PO_DETAIL table by PO_NUMBER:

```
SELECT PO_NUMBER, MIN(QUANTITY), MAX(QUANTITY)
FROM PO_DETAIL
GROUP BY PO_NUMBER;
```

Grouping Data – Cont.

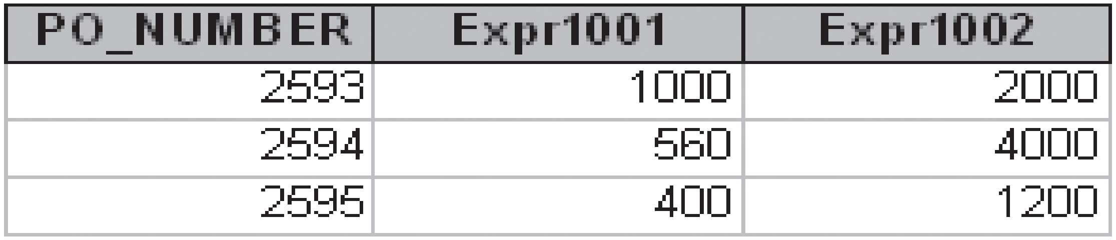


PO NUMBER	Expr1001	Expr1002
2591	300	1000
2592	210.5	800.5
2593	1000	2000
2594	560	4000
2595	400	1200

- A sub-clause of the GROUP BY clause is the **HAVING** clause. This clause allows the programmer to place a condition or filter on the group.

Grouping Data – Cont.

```
SELECT PO_NUMBER, MIN(QUANTITY), MAX(QUANTITY)
FROM PO_DETAIL
GROUP BY PO_NUMBER HAVING MAX(QUANTITY)>1000;
```



PO_NUMBER	Expr1001	Expr1002
2593	1000	2000
2594	560	4000
2595	400	1200

Sub-queries in SQL – Cont.

- Sub-queries allow the user to condition one query on the results of another query from a table

- It also allow the user to retrieve information in a table based on the results of a query in another table

```
Query 1: SELECT AVG(PO_AMT)
FROM PURCHASE_ORDER;
Query 2: SELECT PO_NUMBER, PO_AMT
FROM PURCHASE_ORDER
WHERE PO_AMT > ( result of query 1);
```

Sub-queries in SQL – Cont.

- A command that uses a sub-query structure incorporates both queries in one command:

MAIN QUERY **SELECT <attribute name(s)>**

FROM <TABLE NAME>

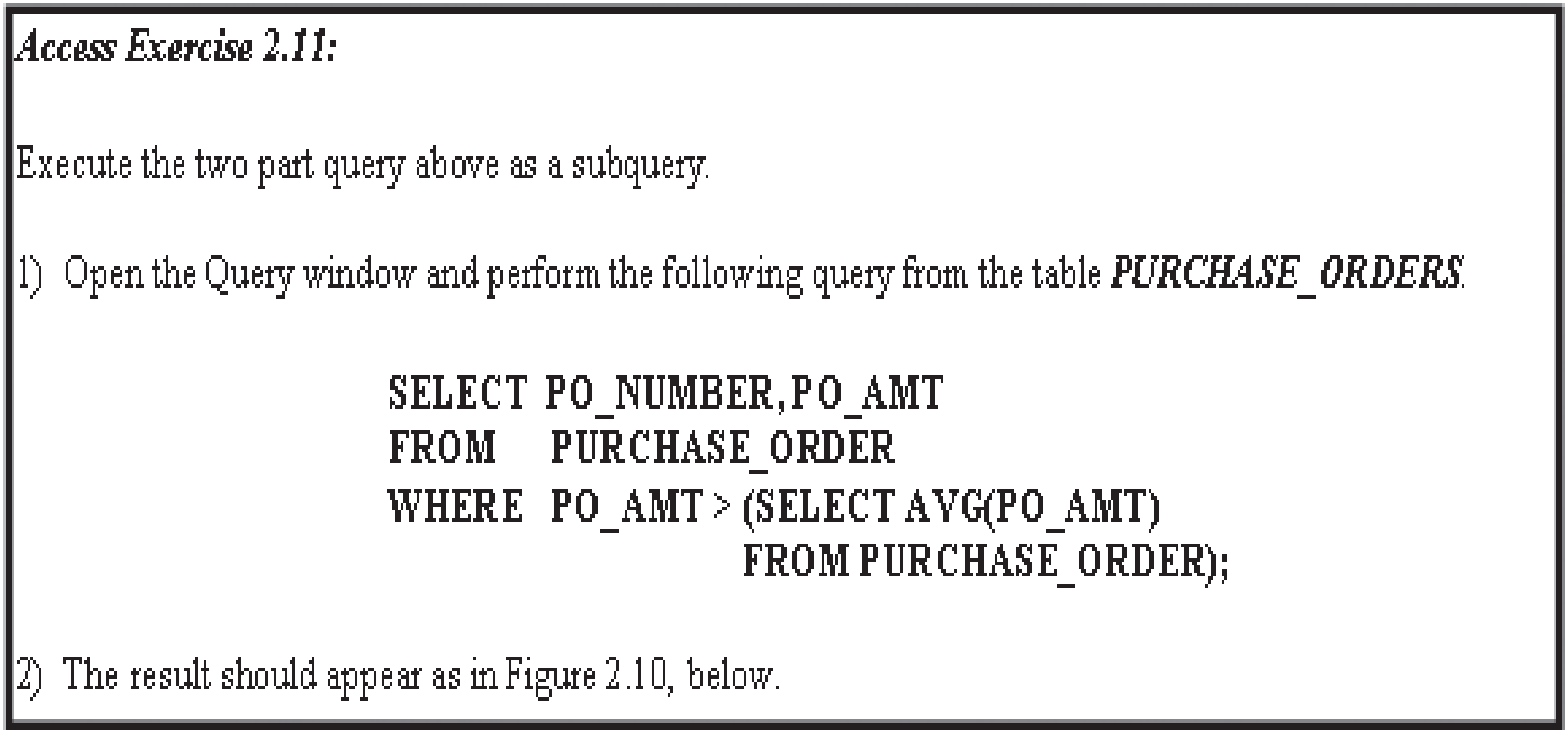
WHERE <column name> <criterion> / <IN>

SUBQUERY **(SELECT <column name>**

FROM <table name>

[WHERE <condition>]);

Sub-queries in SQL – Cont.



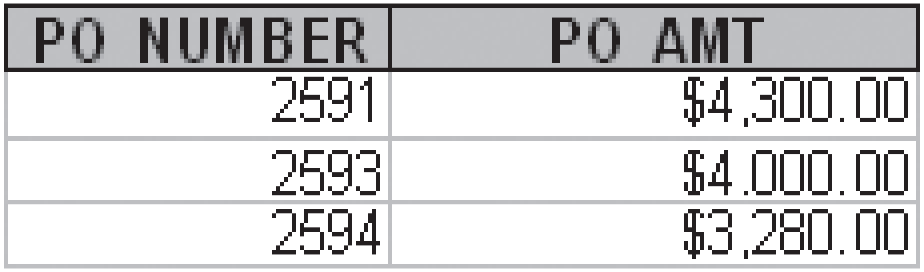
Access Exercise 2.11:

Execute the two part query above as a subquery.

1) Open the Query window and perform the following query from the table ***PURCHASE_ORDERS***.

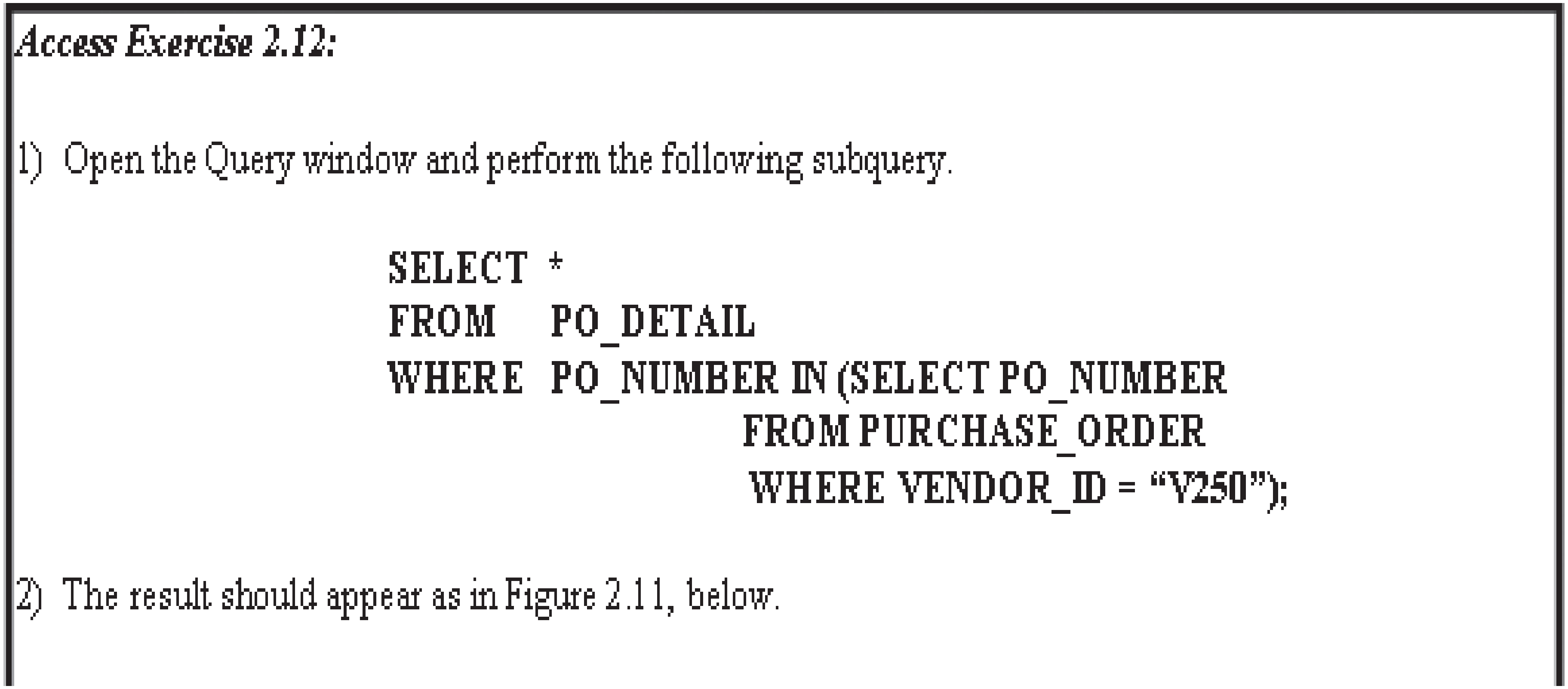
```
SELECT PO_NUMBER,PO_AMT
FROM   PURCHASE_ORDER
WHERE  PO_AMT > (SELECT AVG(PO_AMT)
                 FROM PURCHASE_ORDER);
```

2) The result should appear as in Figure 2.10, below.



PO NUMBER	PO AMT
2591	$4,300.00
2593	$4.000.00
2594	$3,280.00

Sub-queries in SQL – Cont.

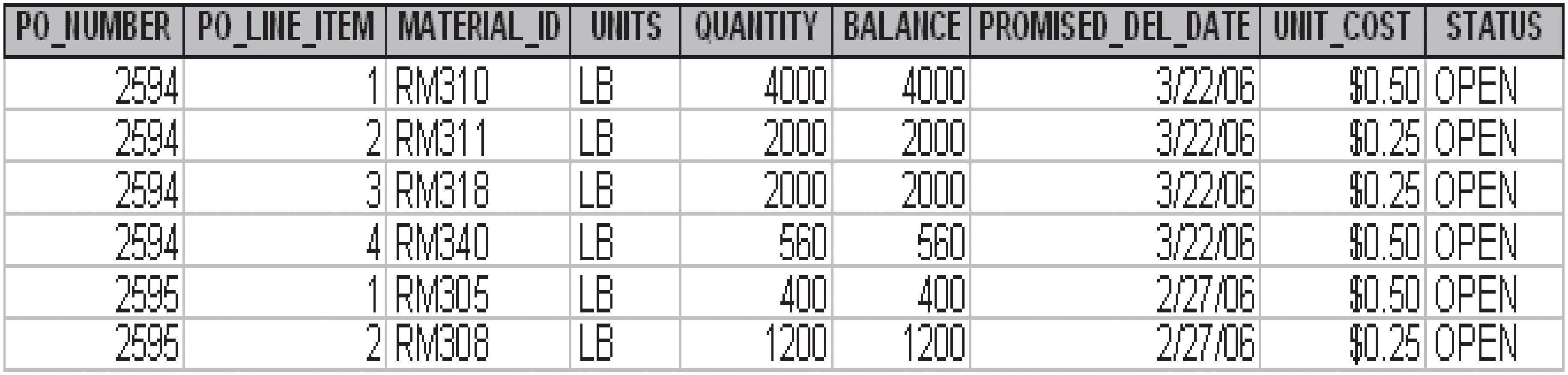


Access Exercise 2.12:

1) Open the Query window and perform the following subquery.

```
SELECT  *
FROM    PO_DETAIL
WHERE   PO_NUMBER IN (SELECT PO_NUMBER
                      FROM PURCHASE_ORDER
                      WHERE VENDOR_ID = “V250”);
```

2) The result should appear as in Figure 2.11, below.



PO_NUMBER	PO_LINE_ITEM	MATERIAL_ID	UNITS	QUANTITY	BALANCE	PROMISED_DEL_DATE	UNIT_COST	STATUS
2594	1	RM310	LB	4000	4000	3/22/06	$0.50	OPEN
2594	2	RM311	LB	2000	2000	3/22/06	$0.25	OPEN
2594	3	RM318	LB	2000	2000	3/22/06	$0.25	OPEN
2594	4	RM340	LB	560	560	3/22/06	$0.50	OPEN
2595	1	RM305	LB	400	400	2/27/06	$0.50	OPEN
2595	2	RM308	LB	1200	1200	2/27/06	$0.25	OPEN

Sub-queries in SQL – Cont.

- The **IN** operator states that the main query is conditioned on the PO_NUMBER(s) that are returned *in the* sub-query

Appending Tables Using Joins

- There are times when it is desirable to display information from more than one table on the same retrieval:

SELECT <table1 name.attribute name>, <table2

name.attribute name>,...

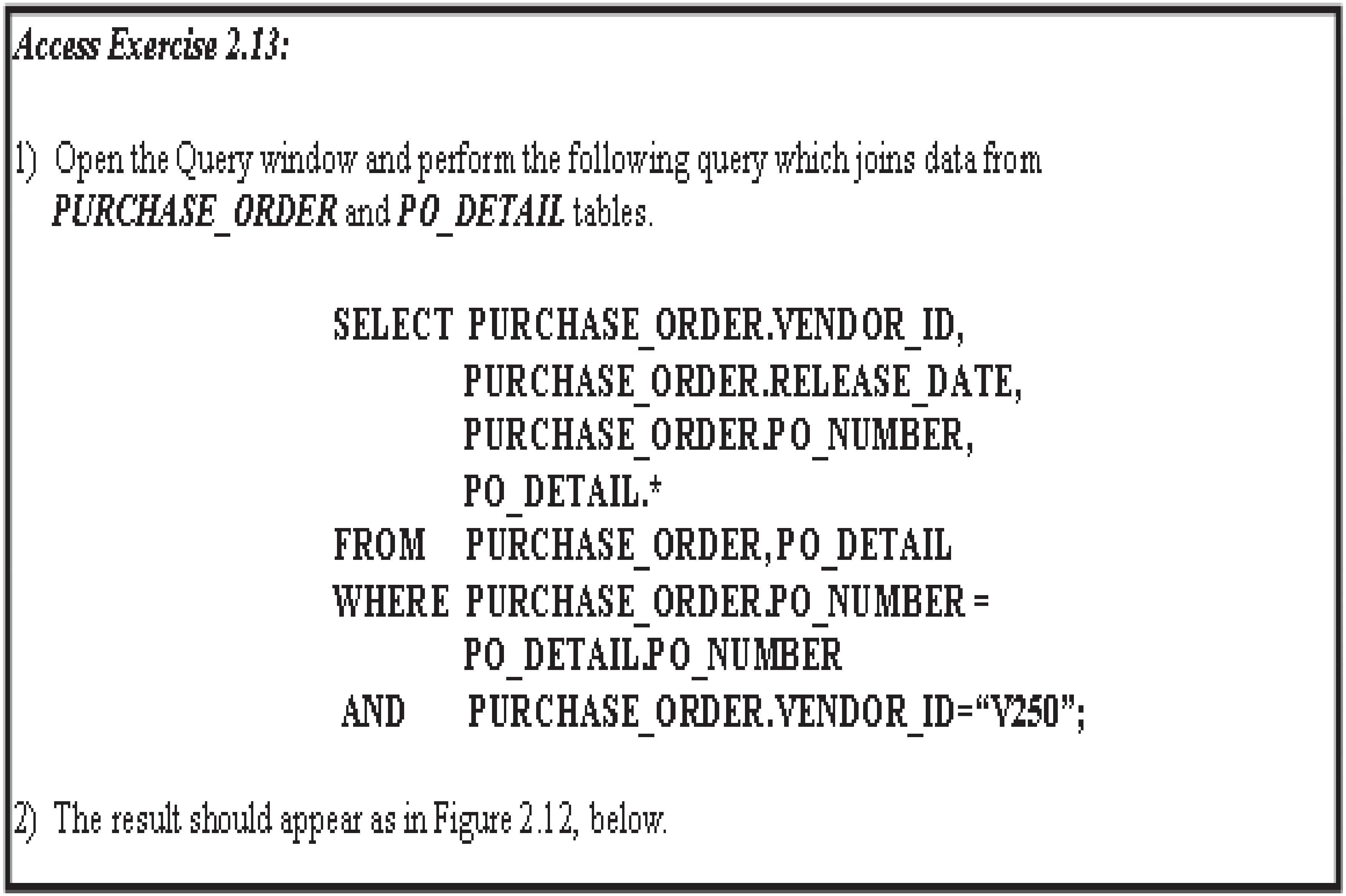
FROM <table1 name>, <table2 name>,...

WHERE <join condition>

ORDER BY <column name>

- For example, one might want to display the VENDOR_ID and RELEASE_DATE from the PURCHASE_ORDER table along with the related details from the PO_DETAIL table. This is the purpose of a table **join.**

Appending Tables Using Joins – Cont.



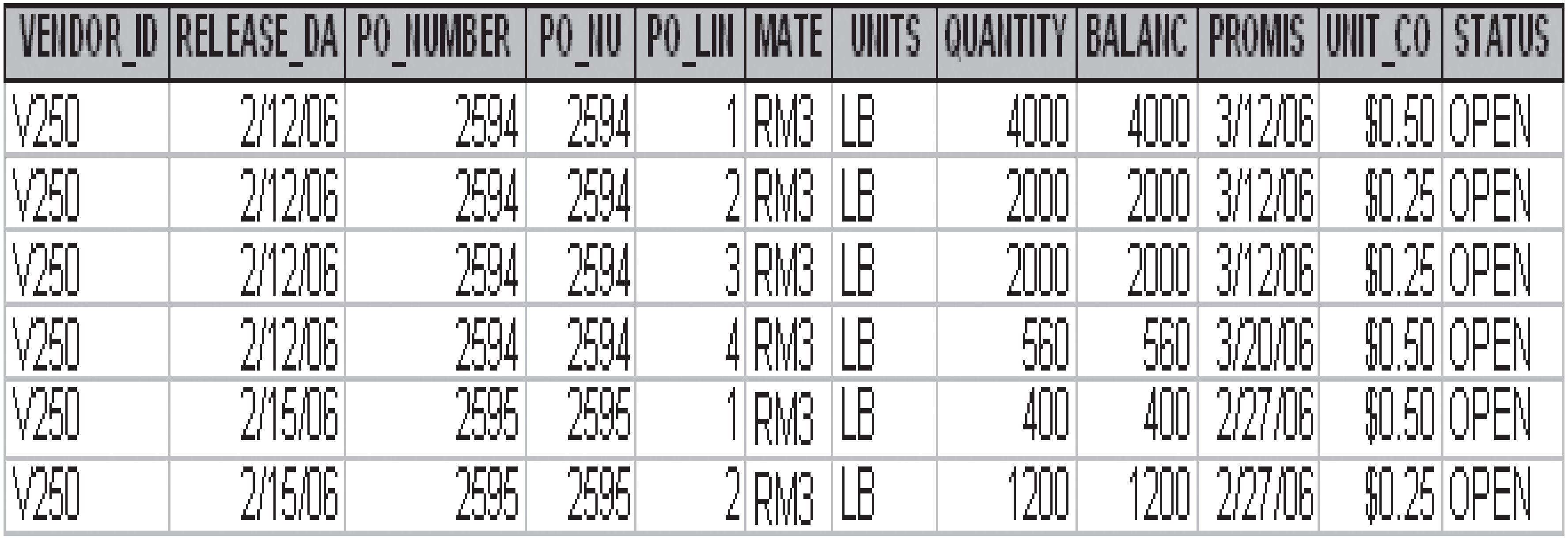
Access Exercise 2.13:

1) Open the Query window and perform the following query which joins data from ***PURCHASE_ORDER*** and ***PO_DETAIL*** tables.

```
SELECT PURCHASE_ORDER.VENDOR_ID,
       PURCHASE_ORDER.RELEASE_DATE,
       PURCHASE_ORDER.PO_NUMBER,
       PO_DETAIL.*
FROM   PURCHASE_ORDER,PO_DETAIL
WHERE  PURCHASE_ORDER.PO_NUMBER =
       PO_DETAIL.PO_NUMBER
 AND   PURCHASE_ORDER.VENDOR_ID=“V250”;
```

2) The result should appear as in Figure 2.12, below.

Appending Tables Using Joins – Cont.

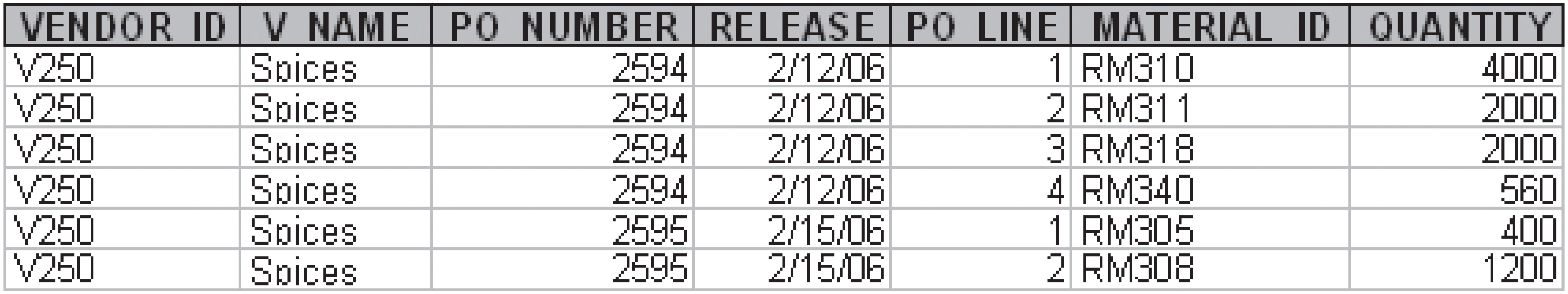


VENDOR_ID	RELEASE_DA	PO_NUMBER	PO_NU	PO_LIN	MATE	UNITS	QUANTITY	BALANC	PROMIS	UNIT_CO	STATUS
V250	2/12/06	2594	2594	1	RM3	LB	4000	4000	3/12/06	$0.50	OPEN
V250	2/12/06	2594	2594	2	RM3	LB	2000	2000	3/12/06	$0.25	OPEN
V250	2/12/06	2594	2594	3	RM3	LB	2000	2000	3/12/06	$0.25	OPEN
V250	2/12/06	2594	2594	4	RM3	LB	560	560	3/20/06	$0.50	OPEN
V250	2/15/06	2595	2595	1	RM3	LB	400	400	2/27/06	$0.50	OPEN
V250	2/15/06	2595	2595	2	RM3	LB	1200	1200	2/27/06	$0.25	OPEN

Appending Tables Using Joins – Cont.

- **Another join Example**:

```
SELECT VENDOR.VENDOR_ID, VENDOR.V_NAME,
PURCHASE_ORDER.PO_NUMBER,
PURCHASE_ORDER.RELEASE_DATE,
PO_DETAIL.PO_LINE_ITEM, PO_DETAIL.MATERIAL_ID,
PO_DETAIL.QUANTITY
FROM VENDOR, PURCHASE_ORDER, PO_DETAIL
WHERE VENDOR.VENDOR_ID = PURCHASE_ORDER.VENDOR_ID
AND PURCHASE_ORDER.PO_NUMBER=PO_DETAIL.PO_NUMBER
AND VENDOR.VENDOR_ID="V250";
```



VENDOR ID	V NAME	PO NUMBER	RELEASE	PO LINE	MATERIAL ID	QUANTITY
V250	Spices	2594	2/12/06	1	RM310	4000
V250	Spices	2594	2/12/06	2	RM311	2000
V250	Spices	2594	2/12/06	3	RM318	2000
V250	Spices	2594	2/12/06	4	RM340	560
V250	Spices	2595	2/15/06	1	RM305	400
V250	Spices	2595	2/15/06	2	RM308	1200