11ed Chapter 7

Introduction to Structured Query Language (SQL)

NOTE

Several points are worth emphasizing:

- We have provided the SQL scripts for both chapters 7 and 8. These scripts are intended to facilitate the flow of the material presented to the class. However, given the comments made by our students, the scripts should **not** replace the manual typing of the SQL commands by students. Some students learn SQL better when they have a chance to type their own commands and get the feedback provided by their errors. We recommend that the students use their lab time to practice the commands manually.
- Because this chapter focuses on learning SQL, we recommend that you use the Microsoft Access SQL window to type SQL queries. Using this approach, you will be able to demonstrate the interoperability of standard SQL. For example, you can cut and paste the same SQL command from the SQL query window in Microsoft Access, to Oracle SQL * Plus and to MS SQL Query Analyzer. This approach achieves two objectives:
  - ➢ It demonstrates that adhering to the SQL standard means that most of the SQL code will be portable among DBMSes.
  - ➢ It also demonstrates that even a widely accepted SQL standard is sometimes implemented with slight distinctions by different vendors. For example, the treatment of date formats in Microsoft Access and Oracle is slightly different.

Answers to Review Questions

1. In a SELECT query, what is the difference between a WHERE clause and a HAVING clause?

Both a WHERE clause and a HAVING clause can be used to eliminate rows from the results of a query. The differences are 1) the WHERE clause eliminates rows before any grouping for aggregate functions occurs while the HAVING clause eliminates groups after the grouping has been done, and 2) the WHERE clause cannot contain an aggregate function but the HAVING clause can.

2. Explain why the following command would create an error, and what changes could be made to fix the error.
SELECT V_CODE, SUM(P_QOH) FROM PRODUCT;

The command would generate an error because an aggregate function is applied to the P_QOH attribute but V_CODE is neither in an aggregate function or in a GROUP BY. This can be fixed by either 1) placing V_CODE in an appropriate aggregate function based on the data that is being requested by the user, 2) adding a GROUP BY clause to group by values of V_CODE (i.e. GROUP BY V_CODE), 3) removing the V_CODE attribute from the SELECT clause, or 4) removing the Sum aggregate function from P_QOH. Which of these solutions is most appropriate depends on the question that the query was intended to answer.

3. **What type of integrity is enforced when a primary key is declared?**

   Creating a primary key constraint enforces **entity integrity** (i.e. no part of the primary key can contain a null and the primary key values must be unique).

4. **Explain why it might be more appropriate to declare an attribute that contains only digits as a character data type instead of a numeric data type.**

   An attribute that contains only digits may be properly defined as character data when the values are nominal; that is, the values do not have numerical significance but serve only as labels such as ZIP codes and telephone numbers. One easy test is to consider whether or not a leading zero should be retained. For the ZIP code 03133, the leading zero should be retained; therefore, it is appropriate to define it as character data. For the quantity on hand of 120, we would not expect to retain a leading zero such as 0120; therefore, it is appropriate to define the quantity on hand as a numeric data type.

5. **What is the difference between a column constraint and a table constraint?**

   A column constraint can refer to only the attribute with which it is specified. A table constraint can refer to any attributes in the table.

6. **What are "referential constraint actions"?**

   Referential constraint actions, such as ON DELETE CASCADE, are default actions that the DBMS should take when a DML command would result in a referential integrity constraint violation. Without referential constraint actions, DML commands that would result in a violation of referential integrity will fail with an error indicating that the referential integrity constrain cannot be violated. Referential constraint actions can allow the DML command to successfully complete while making the designated changes to the related records to maintain referential integrity.

7. **Rewrite the following WHERE clause without the use of the IN special operator.**
   WHERE V_STATE IN ('TN', 'FL', 'GA')

   WHERE V_STATE = 'TN' OR V_STATE = 'FL' OR V_STATE = 'GA'

   Notice that each criteria must be complete (i.e. attribute-operator-value).

8. **Explain the difference between an ORDER BY clause and a GROUP BY clause.**

   An ORDER BY clause has no impact on which rows are returned by the query, it simply sorts those rows into the specified order. A GROUP BY clause does impact the rows that are returned by the query. A GROUP BY clause gathers rows into collections that can be acted on by aggregate functions.

9. **Explain why the two following commands produce different results.**
   SELECT DISTINCT COUNT (V_CODE) FROM PRODUCT;

SELECT COUNT (DISTINCT V_CODE) FROM PRODUCT;

The difference is in the order of operations. The first command executes the Count function to count the number of values in V_CODE (say the count returns "14" for example) including duplicate values, and then the Distinct keyword only allows one count of that value to be displayed (only one row with the value "14" appears as the result). The second command applies the Distinct keyword to the V_CODEs before the count is taken so only unique values are counted.

10. What is the difference between the COUNT aggregate function and the SUM aggregate function?

COUNT returns the number of values without regard to what the values are. SUM adds the values together and can only be applied to numeric values.

11. Explain why it would be preferable to use a DATE data type to store date data instead of a character data type.

The DATE data type uses numeric values based on the Julian calendar to store dates. This makes date arithmetic such as adding and subtracting days or fractions of days possible (as well as numerous special date-oriented functions discussed in the next chapter!).

12. What is a recursive join?

A recursive join is a join in which a table is joined to itself.

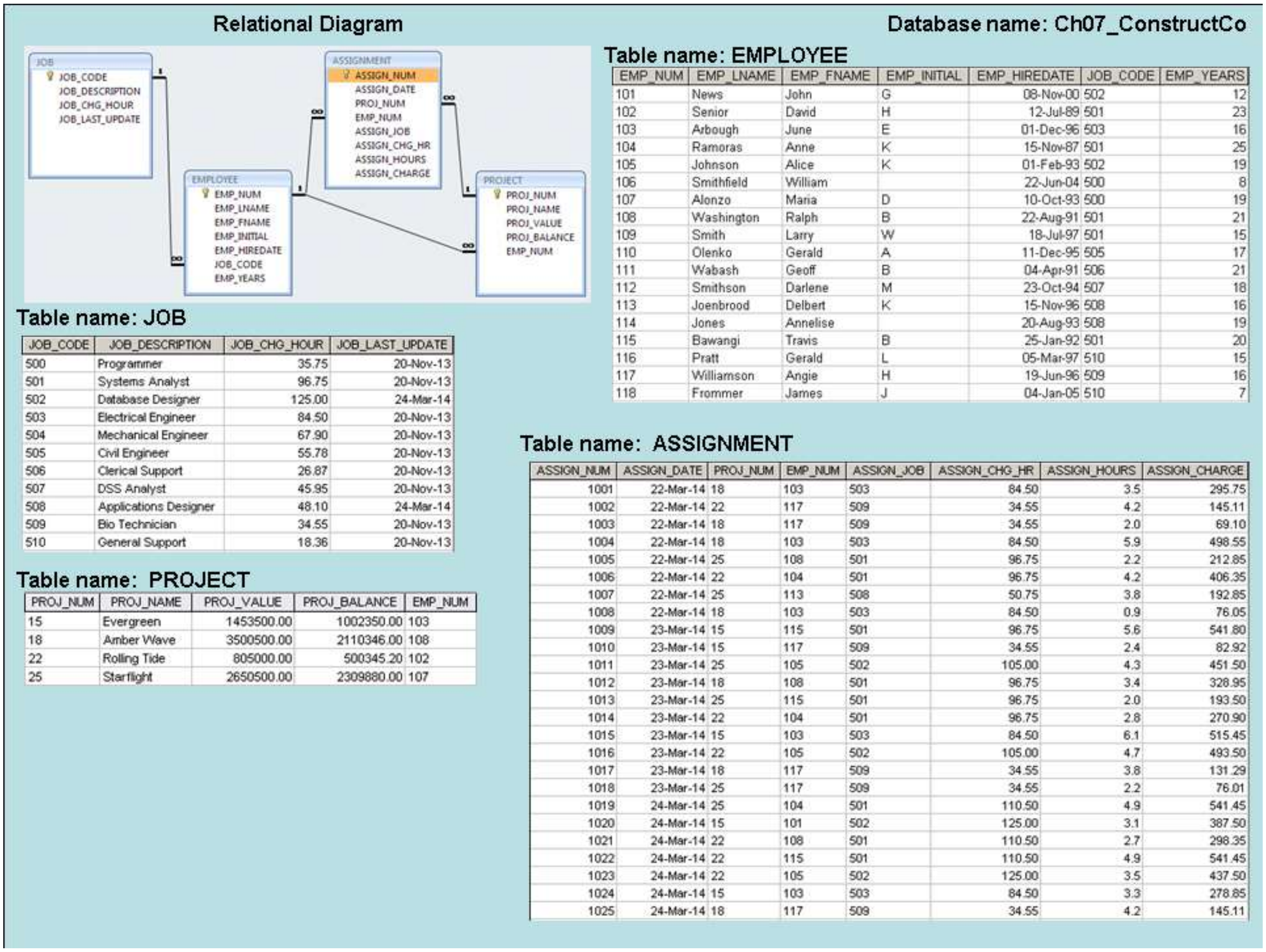
Problem Solutions

Online Content

Problems 1 – 25 are based on the **Ch07_ConstructCo** database located *www.cengagebrain.com*. This database is stored in Microsoft Access format. The website provides Oracle, MySQL, and MS SQL Server script files.

The **Ch07_ConstructCo** database stores data for a consulting company that tracks all charges to projects. The charges are based on the hours each employee works on each project. The structure and contents of the **Ch07_ConstructCo** database are shown in Figure P7.1.

Figure P7.1 Structure and contents of the Ch07_ConstructCo database



Database name: Ch07_ConstructCo

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_YEARS
101	News	John	G	08-Nov-00	502	12
102	Senior	David	H	12-Jul-89	501	23
103	Arbough	June	E	01-Dec-96	503	16
104	Ramoras	Anne	K	15-Nov-87	501	25
105	Johnson	Alice	K	01-Feb-93	502	19
106	Smithfield	William		22-Jun-04	500	8
107	Alonzo	Maria	D	10-Oct-93	500	19
108	Washington	Ralph	B	22-Aug-91	501	21
109	Smith	Larry	W	18-Jul-97	501	15
110	Olenko	Gerald	A	11-Dec-95	505	17
111	Wabash	Geoff	B	04-Apr-91	506	21
112	Smithson	Darlene	M	23-Oct-94	507	18
113	Joenbrood	Delbert	K	15-Nov-96	508	16
114	Jones	Annelise		20-Aug-93	508	19
115	Bawangi	Travis	B	25-Jan-92	501	20
116	Pratt	Gerald	L	05-Mar-97	510	15
117	Williamson	Angie	H	19-Jun-96	509	16
118	Frommer	James	J	04-Jan-05	510	7

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR	JOB_LAST_UPDATE
500	Programmer	35.75	20-Nov-13
501	Systems Analyst	96.75	20-Nov-13
502	Database Designer	125.00	24-Mar-14
503	Electrical Engineer	84.50	20-Nov-13
504	Mechanical Engineer	67.90	20-Nov-13
505	Civil Engineer	55.78	20-Nov-13
506	Clerical Support	26.87	20-Nov-13
507	DSS Analyst	45.95	20-Nov-13
508	Applications Designer	48.10	24-Mar-14
509	Bio Technician	34.55	20-Nov-13
510	General Support	18.36	20-Nov-13

Table name: PROJECT

PROJ_NUM	PROJ_NAME	PROJ_VALUE	PROJ_BALANCE	EMP_NUM
15	Evergreen	1453500.00	1002350.00	103
18	Amber Wave	3500500.00	2110346.00	108
22	Rolling Tide	805000.00	500345.20	102
25	Starflight	2650500.00	2309880.00	107

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_JOB	ASSIGN_CHG_HR	ASSIGN_HOURS	ASSIGN_CHARGE
1001	22-Mar-14	18	103	503	84.50	3.5	295.75
1002	22-Mar-14	22	117	509	34.55	4.2	145.11
1003	22-Mar-14	18	117	509	34.55	2.0	69.10
1004	22-Mar-14	18	103	503	84.50	5.9	498.55
1005	22-Mar-14	25	108	501	96.75	2.2	212.85
1006	22-Mar-14	22	104	501	96.75	4.2	406.35
1007	22-Mar-14	25	113	508	50.75	3.8	192.85
1008	22-Mar-14	18	103	503	84.50	0.9	76.05
1009	23-Mar-14	15	115	501	96.75	5.6	541.80
1010	23-Mar-14	15	117	509	34.55	2.4	82.92
1011	23-Mar-14	25	105	502	105.00	4.3	451.50
1012	23-Mar-14	18	108	501	96.75	3.4	328.95
1013	23-Mar-14	25	115	501	96.75	2.0	193.50
1014	23-Mar-14	22	104	501	96.75	2.8	270.90
1015	23-Mar-14	15	103	503	84.50	6.1	515.45
1016	23-Mar-14	22	105	502	105.00	4.7	493.50
1017	23-Mar-14	18	117	509	34.55	3.8	131.29
1018	23-Mar-14	25	117	509	34.55	2.2	76.01
1019	24-Mar-14	25	104	501	110.50	4.9	541.45
1020	24-Mar-14	15	101	502	125.00	3.1	387.50
1021	24-Mar-14	22	108	501	110.50	2.7	298.35
1022	24-Mar-14	22	115	501	110.50	4.9	541.45
1023	24-Mar-14	22	105	502	125.00	3.5	437.50
1024	24-Mar-14	15	103	503	84.50	3.3	278.85
1025	24-Mar-14	18	117	509	34.55	4.2	145.11

Note that the ASSIGNMENT table in Figure P7.1 stores the JOB_CHG_HOUR values as an attribute (ASSIGN_CHG_HR) to maintain historical accuracy of the data. The JOB_CHG_HOUR values are likely to change over time. In fact, a JOB_CHG_HOUR change will be reflected in the ASSIGNMENT table. And, naturally, the employee primary job assignment might change, so the ASSIGN_JOB is also stored. Because those attributes are required to maintain the historical accuracy of the data, they are *not* redundant.

Given the structure and contents of the Ch07_ConstructCo **database shown in Figure P7.1, use SQL commands to answer Problems 1–25.**

1. **Write the SQL code that will create the table structure for a table named EMP_1. This table is a subset of the EMPLOYEE table. The basic EMP_1 table structure is summarized in the table below. (Note that the JOB_CODE is the FK to JOB.)**

ATTRIBUTE (FIELD) NAME	DATA DECLARATION
EMP_NUM	CHAR(3)
EMP_LNAME	VARCHAR(15)
EMP_FNAME	VARCHAR(15)
EMP_INITIAL	CHAR(1)
EMP_HIREDATE	DATE
JOB_CODE	CHAR(3)

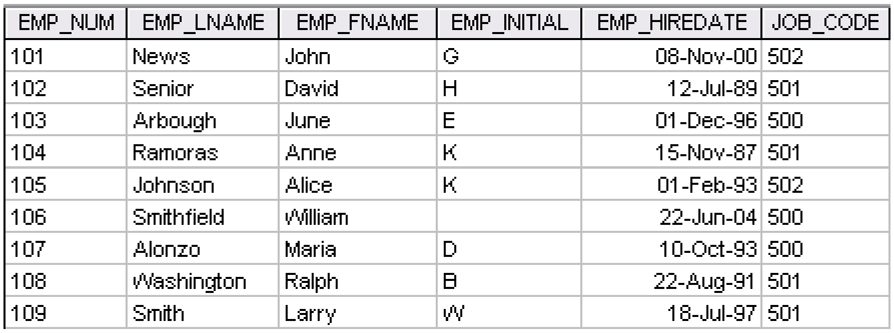
```
 CREATE TABLE EMP_1 (
 EMP_NUM             CHAR(3)            PRIMARY KEY,
 EMP_LNAME           VARCHAR(15)        NOT NULL,
 EMP_FNAME           VARCHAR(15)        NOT NULL,
 EMP_INITIAL         CHAR(1),
 EMP_HIREDATE        DATE,
 JOB_CODE            CHAR(3),
FOREIGN KEY (JOB_CODE) REFERENCES JOB);
```

NOTE

We have already provided the EMP_1 table for you. If you try to run the preceding query, you will get an error message because the EMP_1 table already exits.

2. Having created the table structure in Problem 1, write the SQL code to enter the first two rows for the table shown in Figure P7.2.

Figure P7.2 The contents of the EMP_1 table



EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-96	500
104	Ramoras	Anne	K	15-Nov-87	501
105	Johnson	Alice	K	01-Feb-93	502
106	Smithfield	William		22-Jun-04	500
107	Alonzo	Maria	D	10-Oct-93	500
108	Washington	Ralph	B	22-Aug-91	501
109	Smith	Larry	W	18-Jul-97	501

```
 INSERT INTO EMP_1 VALUES (‘101’, ‘News’, ‘John’, ‘G’, ’08-Nov-00’, ‘502’);
 INSERT INTO EMP_1 VALUES (‘102’, ‘Senior’, ‘David’, ‘H’, ’12-Jul-89’, ‘501’);
```

3. Assuming the data shown in the EMP_1 table have been entered, write the SQL code that will list all attributes for a job code of 502.

```
SELECT    *
FROM      EMP_1
WHERE     JOB_CODE = ‘502’;
```

4. Write the SQL code that will save the changes made to the EMP_1 table.

```
COMMIT;
```

5. Write the SQL code to change the job code to 501 for the person whose employee number (EMP_NUM) is 107. After you have completed the task, examine the results, and then reset the job code to its original value.

```
UPDATE EMP_1
SET      JOB_CODE = ‘501’
WHERE    EMP_NUM = ‘107’;
```

To see the changes:

```
SELECT   *
FROM     EMP_1
WHERE    EMP_NUM = ‘107’;
```

To reset, use

```
ROLLBACK;
```

6. Write the SQL code to delete the row for the person named William Smithfield, who was hired on June 22, 2004, and whose job code classification is 500. (*Hint*: Use logical operators to include all of the information given in this problem.)

```
DELETE   FROM EMP_1
WHERE    EMP_LNAME = 'Smithfield'
AND      EMP_FNAME = 'William'
AND      EMP_HIREDATE = '22-June-04'
AND      JOB_CODE = '500';
```

7. Write the SQL code that will restore the data to its original status; that is, the table should contain the data that existed before you made the changes in Problems 5 and 6.

```
ROLLBACK;
```

8. **Write the SQL code to create a copy of EMP_1, naming the copy EMP_2. Then write the SQL code that will add the attributes EMP_PCT and PROJ_NUM to its structure. The EMP_PCT is the bonus percentage to be paid to each employee. The new attribute characteristics are:**

   EMP_PCTNUMBER(4,2)

   PROJ_NUMCHAR(3)

   (*Note*: If your SQL implementation allows it, you may use DECIMAL(4,2) rather than NUMBER(4,2).)

There are two way to get this job done. The two possible solutions are shown next.

Solution A:

```
CREATE TABLE EMP_2 (
        EMP_NUM             CHAR(3)           NOT NULL UNIQUE,
        EMP_LNAME           VARCHAR(15)       NOT NULL,
        EMP_FNAME           VARCHAR(15)       NOT NULL,
        EMP_INITIAL         CHAR(1),
        EMP_HIREDATE        DATE              NOT NULL,
        JOB_CODE            CHAR(3)           NOT NULL,
        PRIMARY KEY (EMP_NUM),
        FOREIGN KEY (JOB_CODE) REFERENCES JOB);

INSERT INTO EMP_2 SELECT * FROM EMP_1;

 ALTER TABLE EMP_2
    ADD (EMP_PCT       NUMBER (4,2)),
    ADD (PROJ_NUM      CHAR(3));
```

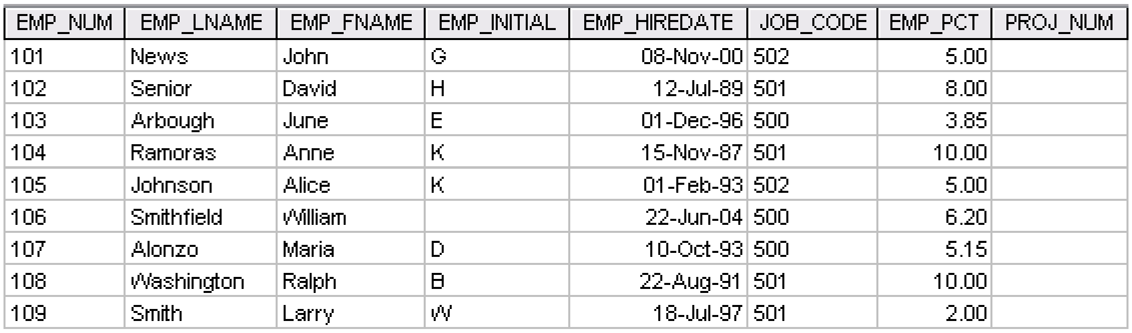
Solution B:

```
CREATE TABLE EMP_2 AS SELECT * FROM EMP_1;

ALTER TABLE EMP_2
    ADD (EMP_PCT       NUMBER (4,2)),
    ADD (PROJ_NUM      CHAR(3));
```

9. **Write the SQL code to change the EMP_PCT value to 3.85 for the person whose employee number (EMP_NUM) is 103. Next, write the SQL command sequences to change the EMP_PCT values as shown in Figure P7.9.**

Figure P7.9 The contents of the EMP_2 table



EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
101	News	John	G	08-Nov-00	502	5.00	
102	Senior	David	H	12-Jul-89	501	8.00	
103	Arbough	June	E	01-Dec-96	500	3.85	
104	Ramoras	Anne	K	15-Nov-87	501	10.00	
105	Johnson	Alice	K	01-Feb-93	502	5.00	
106	Smithfield	William		22-Jun-04	500	6.20	
107	Alonzo	Maria	D	10-Oct-93	500	5.15	
108	Washington	Ralph	B	22-Aug-91	501	10.00	
109	Smith	Larry	W	18-Jul-97	501	2.00	

```
UPDATE  EMP_2
SET     EMP_PCT = 3.85
WHERE   EMP_NUM = '103';
```

To enter the remaining EMP_PCT values, use the following SQL statements:

```
UPDATE EMP_2
SET    EMP_PCT = 5.00
WHERE  EMP_NUM = ‘101’;

UPDATE EMP_2
SET    EMP_PCT = 8.00
WHERE  EMP_NUM = ‘102’;
```

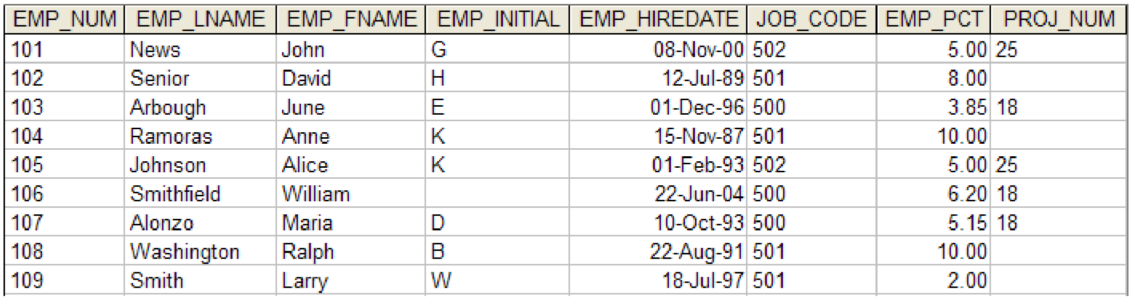
Follow this format for the remaining rows.

10. Using a single command sequence, write the SQL code that will change the project number (PROJ_NUM) to 18 for all employees whose job classification (JOB_CODE) is 500.

```
UPDATE  EMP_2
SET     PROJ_NUM = '18'
WHERE   JOB_CODE = '500';
```

11. Using a single command sequence, write the SQL code that will change the project number (PROJ_NUM) to 25 for all employees whose job classification (JOB_CODE) is 502 or higher. When you finish Problems 10 and 11, the EMP_2 table will contain the data shown in Figure P7.11. (You may assume that the table has been saved again at this point.)

Figure P7.11 The EMP_2 table contents after the modification



EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
101	News	John	G	08-Nov-00	502	5.00	25
102	Senior	David	H	12-Jul-89	501	8.00	
103	Arbough	June	E	01-Dec-96	500	3.85	18
104	Ramoras	Anne	K	15-Nov-87	501	10.00	
105	Johnson	Alice	K	01-Feb-93	502	5.00	25
106	Smithfield	William		22-Jun-04	500	6.20	18
107	Alonzo	Maria	D	10-Oct-93	500	5.15	18
108	Washington	Ralph	B	22-Aug-91	501	10.00	
109	Smith	Larry	W	18-Jul-97	501	2.00	

```
UPDATE  EMP_2
SET     PROJ_NUM = '25'
WHERE   JOB_CODE > = '502'
```

12. Write the SQL code that will change the PROJ_NUM to 14 for those employees who were hired before January 1, 1994 and whose job code is at least 501. (You may assume that the table will be restored to its condition preceding this question.)

```
UPDATE  EMP_2
SET     PROJ_NUM = '14'
WHERE   EMP_HIREDATE <= ' 01-Jan-94'
AND     JOB_CODE >= '501';
```

13. Write the two SQL command sequences required to:

There are many ways to accomplish both tasks. We are illustrating the shortest way to do the job next.

a. Create a temporary table named TEMP_1 whose structure is composed of the EMP_2 attributes EMP_NUM and EMP_PCT.

The SQL code shown in problem 13b contains the solution for problem 13a.

b. Copy the matching EMP_2 values into the TEMP_1 table.

```
CREATE TABLE TEMP_1 AS SELECT EMP_NUM, EMP_PCT FROM EMP_2;
```

An alternate way would be to create the table and then, use an INSERT with a sub-select to populate the rows.

```
CREATE TABLE TEMP_1 AS (
EMP_NUM     CHAR(3),
EMP_PCT     NUMBER(4,2));

INSERT INTO TEMP_1
SELECT EMP_NUM, EMP_PCT FROM EMP_2;
```

14. Write the SQL command that will delete the newly created TEMP_1 table from the database.

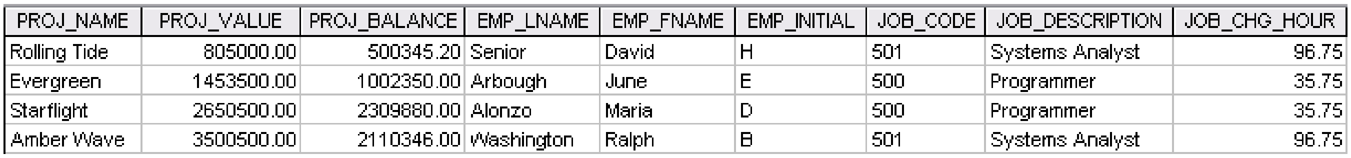
```
DROP TABLE TEMP_1;
```

15. Write the SQL code required to list all employees whose last names start with *Smith*. In other words, the rows for both Smith and Smithfield should be included in the listing. Assume case sensitivity.

```
SELECT   *
FROM     EMP_2
WHERE    EMP_LNAME LIKE 'Smith%';
```

16. Using the EMPLOYEE, JOB, and PROJECT tables in the Ch07_ConstructCo database (see Figure P7.1), write the SQL code that will produce the results shown in Figure P7.16.

Figure P7.16 The query results for Problem 16



PROJ_NAME	PROJ_VALUE	PROJ_BALANCE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
Rolling Tide	805000.00	500345.20	Senior	David	H	501	Systems Analyst	96.75
Evergreen	1453500.00	1002350.00	Arbough	June	E	500	Programmer	35.75
Starflight	2650500.00	2309880.00	Alonzo	Maria	D	500	Programmer	35.75
Amber Wave	3500500.00	2110346.00	Washington	Ralph	B	501	Systems Analyst	96.75

```
SELECT    PROJ_NAME, PROJ_VALUE, PROJ_BALANCE, EMPLOYEE.EMP_LNAME,
          EMP_FNAME, EMP_INITIAL, EMPLOYEE.JOB_CODE, JOB.JOB_DESCRIPTION,
          JOB.JOB_CHG_HOUR
 FROM     PROJECT, EMPLOYEE, JOB
 WHERE    EMPLOYEE.EMP_NUM = PROJECT.EMP_NUM
 AND      JOB.JOB_CODE = EMPLOYEE.JOB_CODE;
```

17. Write the SQL code that will produce a virtual table named REP_1. The virtual table should contain the same information that was shown in Problem 16.

```
CREATE VIEW REP_1 AS
SELECT   PROJ_NAME, PROJ_VALUE, PROJ_BALANCE, EMPLOYEE.EMP_LNAME,
         EMP_FNAME, EMP_INITIAL, EMPLOYEE.JOB_CODE, JOB.JOB_DESCRIPTION,
         JOB.JOB_CHG_HOUR
 FROM    PROJECT, EMPLOYEE, JOB
 WHERE   EMPLOYEE.EMP_NUM = PROJECT.EMP_NUM
 AND     JOB.JOB_CODE = EMPLOYEE.JOB_CODE;
```

18. Write the SQL code to find the average bonus percentage in the EMP_2 table you created in Problem 8.

```
SELECT   AVG(EMP_PCT)
FROM     EMP_2;
```

19. Write the SQL code that will produce a listing for the data in the EMP_2 table in ascending order by the bonus percentage.

```
SELECT      *
FROM        EMP_2
ORDER BY    EMP_PCT;
```

20. Write the SQL code that will list only the distinct project numbers found in the EMP_2 table.

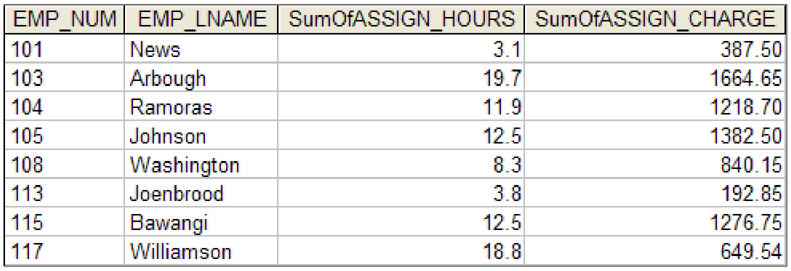
```
SELECT   DISTINTC PROJ_NUM
FROM     EMP_2;
```

21. Write the SQL code to calculate the ASSIGN_CHARGE values in the ASSIGNMENT table in the Ch07_ConstructCo database. (See Figure P7.1.) Note that ASSIGN_CHARGE is a derived attribute that is calculated by multiplying ASSIGN_CHG_HR by ASSIGN_HOURS.

```
UPDATE ASSIGNMENT
SET ASSIGN_CHARGE = ASSIGN_CHG_HR * ASSIGN_HOURS;
```

22. Using the data in the ASSIGNMENT table, write the SQL code that will yield the total number of hours worked for each employee and the total charges stemming from those hours worked. The results of running that query are shown in Figure P7.22.

Figure P7.22 Total hours and charges by employee

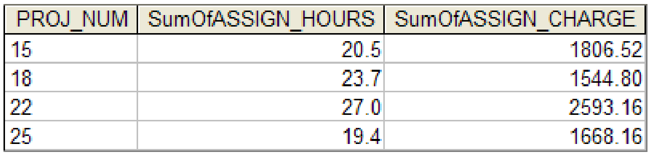


EMP_NUM	EMP_LNAME	SumOfASSIGN_HOURS	SumOfASSIGN_CHARGE
101	News	3.1	387.50
103	Arbough	19.7	1664.65
104	Ramoras	11.9	1218.70
105	Johnson	12.5	1382.50
108	Washington	8.3	840.15
113	Joenbrood	3.8	192.85
115	Bawangi	12.5	1276.75
117	Williamson	18.8	649.54

```
SELECT      ASSIGNMENT.EMP_NUM, EMPLOYEE.EMP_LNAME,
            Sum(ASSIGNMENT.ASSIGN_HOURS) AS SumOfASSIGN_HOURS,
            Sum(ASSIGNMENT.ASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM        EMPLOYEE, ASSIGNMENT
WHERE       EMPLOYEE.EMP_NUM = ASSIGNMENT.EMP_NUM
GROUP BY    ASSIGNMENT.EMP_NUM, EMPLOYEE.EMP_LNAME;
```

23. Write a query to produce the total number of hours and charges for each of the projects represented in the ASSIGNMENT table. The output is shown in Figure P7.23.

Figure P7.23 Total hour and charges by project

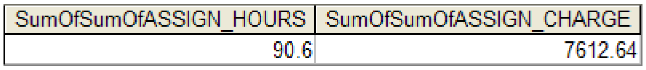


PROJ_NUM	SumOfASSIGN_HOURS	SumOfASSIGN_CHARGE
15	20.5	1806.52
18	23.7	1544.80
22	27.0	2593.16
25	19.4	1668.16

```
SELECT      ASSIGNMENT.PROJ_NUM,
            Sum(ASSIGNMENT.ASSIGN_HOURS) AS SumOfASSIGN_HOURS,
            Sum(ASSIGNMENT.ASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM        ASSIGNMENT
GROUP BY    ASSIGNMENT.PROJ_NUM
```

24. Write the SQL code to generate the total hours worked and the total charges made by all employees. The results are shown in Figure P7.24. (*Hint:* This is a nested query. If you use Microsoft Access, you can generate the result by using the query output shown in Figure P7.22 as the basis for the query that will produce the output shown in Figure P7.24.)

Figure P7.24 Total hours and charges, all employees



SumOfSumOfASSIGN_HOURS	SumOfSumOfASSIGN_CHARGE
90.6	7612.64

Solution A:

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM     Q23;
```

or

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE as SumOfASSIGN_CHARGE
 FROM    (SELECT     ASSIGNMENT.PROJ_NUM,
                     Sum(ASSIGNMENT.ASSIGN_HOURS) AS SumOfASSIGN_HOURS,
                     Sum(ASSIGNMENT.ASSIGN_CHARGE) AS
                          SumOfASSIGN_CHARGE
         FROM        ASSIGNMENT
         GROUP BY    ASSIGNMENT.PROJ_NUM
         );
```

Solution B:

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM     Q22;
```

or

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM         (SELECT  ASSIGNMENT.EMP_NUM, EMPLOYEE.EMP_LNAME,
                      Sum(ASSIGNMENT.ASSIGN_HOURS) AS SumOfASSIGN_HOURS,
                      Sum(ASSIGNMENT.ASSIGN_CHARGE) AS
                           SumOfASSIGN_CHARGE
             FROM     EMPLOYEE, ASSIGNMENT
             WHERE    EMPLOYEE.EMP_NUM = ASSIGNMENT.EMP_NUM
             GROUP BY      ASSIGNMENT.EMP_NUM, EMPLOYEE.EMP_LNAME
             );
```

25. Write the SQL code to generate the total hours worked and the total charges made to all projects. The results should be the same as those shown in Figure P7.24. (*Hint:* This is a nested query. If you use Microsoft Access, you can generate the result by using the query output shown in Figure P7.23 as the basis for this query.)

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE) AS SumOfASSIGN_CHARGE
FROM     Q23;
```

or

```
SELECT   Sum(SumOfASSIGN_HOURS) AS SumOfASSIGN_HOURS,
         Sum(SumOfASSIGN_CHARGE as SumOfASSIGN_CHARGE
 FROM    (SELECT      ASSIGNMENT.PROJ_NUM,
                      Sum(ASSIGNMENT.ASSIGN_HOURS) AS SumOfASSIGN_HOURS,
                      Sum(ASSIGNMENT.ASSIGN_CHARGE) AS
                              SumOfASSIGN_CHARGE
         FROM         ASSIGNMENT
         GROUP BY     ASSIGNMENT.PROJ_NUM
         );
```

Online Content

Problems 26−43 are based on the **Ch07_SaleCo** database, which is available at *www.cengagebrain.com*. This database is stored in Microsoft Access format. Oracle, MySQL and MS SQL Server script files are available at *www.cengagebrain.com*.

The structure and contents of the **Ch07_SaleCo** database are shown in Figure P7.26. Use this database to answer the following problems. Save each query as QXX, where XX is the problem number.

26. Write a query to count the number of invoices.

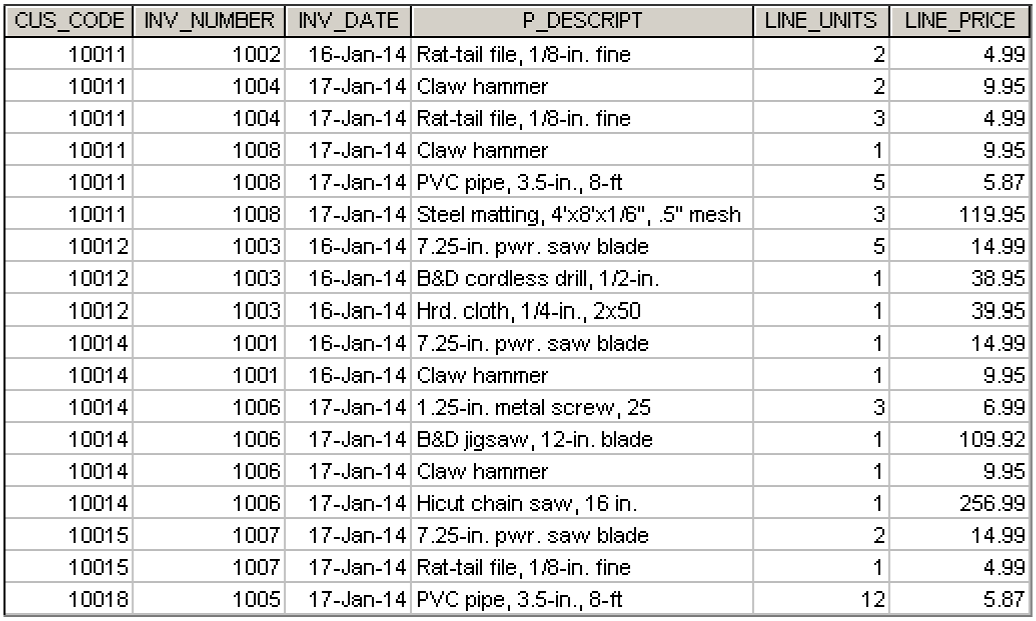
```
SELECT COUNT(*) FROM INVOICE;
```

27. Write a query to count the number of customers with a customer balance over $500.

```
SELECT   COUNT(*)
FROM     CUSTOMER
WHERE    CUS_BALANCE >500;
```

28. Generate a listing of all purchases made by the customers, using the output shown in Figure P7.28 as your guide. (*Hint*: Use the ORDER BY clause to order the resulting rows as shown in Figure P7.28)

FIGURE P7.28 List of Customer Purchases

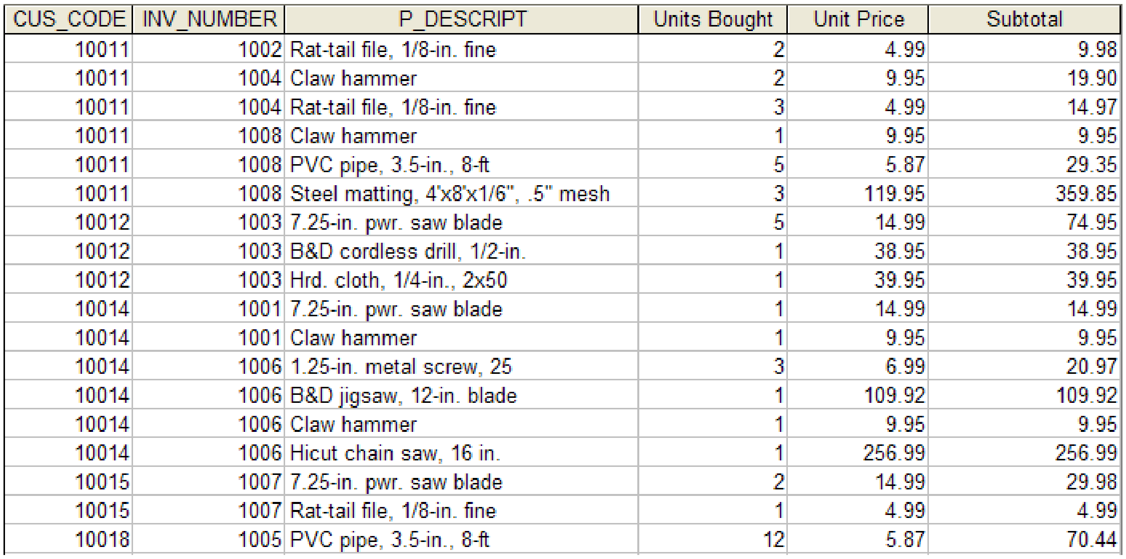


CUS_CODE	INV_NUMBER	INV_DATE	P_DESCRIPT	LINE_UNITS	LINE_PRICE
10011	1002	16-Jan-14	Rat-tail file, 1/8-in. fine	2	4.99
10011	1004	17-Jan-14	Claw hammer	2	9.95
10011	1004	17-Jan-14	Rat-tail file, 1/8-in. fine	3	4.99
10011	1008	17-Jan-14	Claw hammer	1	9.95
10011	1008	17-Jan-14	PVC pipe, 3.5-in., 8-ft	5	5.87
10011	1008	17-Jan-14	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95
10012	1003	16-Jan-14	7.25-in. pwr. saw blade	5	14.99
10012	1003	16-Jan-14	B&D cordless drill, 1/2-in.	1	38.95
10012	1003	16-Jan-14	Hrd. cloth, 1/4-in., 2x50	1	39.95
10014	1001	16-Jan-14	7.25-in. pwr. saw blade	1	14.99
10014	1001	16-Jan-14	Claw hammer	1	9.95
10014	1006	17-Jan-14	1.25-in. metal screw, 25	3	6.99
10014	1006	17-Jan-14	B&D jigsaw, 12-in. blade	1	109.92
10014	1006	17-Jan-14	Claw hammer	1	9.95
10014	1006	17-Jan-14	Hicut chain saw, 16 in.	1	256.99
10015	1007	17-Jan-14	7.25-in. pwr. saw blade	2	14.99
10015	1007	17-Jan-14	Rat-tail file, 1/8-in. fine	1	4.99
10018	1005	17-Jan-14	PVC pipe, 3.5-in., 8-ft	12	5.87

```
SELECT     INVOICE.CUS_CODE, INVOICE.INV_NUMBER, INVOICE.INV_DATE,
           PRODUCT.P_DESCRIPT, LINE.LINE_UNITS, LINE.LINE_PRICE
FROM       CUSTOMER, INVOICE, LINE, PRODUCT
WHERE      CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND        INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND        PRODUCT.P_CODE = LINE.P_CODE
ORDER BY   INVOICE.CUS_CODE, INVOICE.INV_NUMBER, PRODUCT.P_DESCRIPT;
```

29. Using the output shown in Figure P7.29 as your guide, generate the listing of customer purchases, including the subtotals for each of the invoice line numbers. (*Hint*: Modify the query format used to produce the listing of customer purchases in Problem 18, delete the INV_DATE column, and add the derived (computed) attribute LINE_UNITS * LINE_PRICE to calculate the subtotals.)

FIGURE P7.29 Summary of Customer Purchases with Subtotals

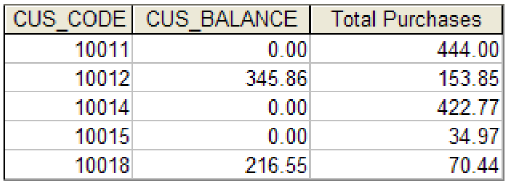


CUS_CODE	INV_NUMBER	P_DESCRIPT	Units Bought	Unit Price	Subtotal
10011	1002	Rat-tail file, 1/8-in. fine	2	4.99	9.98
10011	1004	Claw hammer	2	9.95	19.90
10011	1004	Rat-tail file, 1/8-in. fine	3	4.99	14.97
10011	1008	Claw hammer	1	9.95	9.95
10011	1008	PVC pipe, 3.5-in., 8-ft	5	5.87	29.35
10011	1008	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95	359.85
10012	1003	7.25-in. pwr. saw blade	5	14.99	74.95
10012	1003	B&D cordless drill, 1/2-in.	1	38.95	38.95
10012	1003	Hrd. cloth, 1/4-in., 2x50	1	39.95	39.95
10014	1001	7.25-in. pwr. saw blade	1	14.99	14.99
10014	1001	Claw hammer	1	9.95	9.95
10014	1006	1.25-in. metal screw, 25	3	6.99	20.97
10014	1006	B&D jigsaw, 12-in. blade	1	109.92	109.92
10014	1006	Claw hammer	1	9.95	9.95
10014	1006	Hicut chain saw, 16 in.	1	256.99	256.99
10015	1007	7.25-in. pwr. saw blade	2	14.99	29.98
10015	1007	Rat-tail file, 1/8-in. fine	1	4.99	4.99
10018	1005	PVC pipe, 3.5-in., 8-ft	12	5.87	70.44

```
SELECT      INVOICE.CUS_CODE, INVOICE.INV_NUMBER, PRODUCT.P_DESCRIPT,
            LINE.LINE_UNITS AS [Units Bought], LINE.LINE_PRICE AS [Unit Price],
            LINE.LINE_UNITS*LINE.LINE_PRICE AS Subtotal
FROM        CUSTOMER, INVOICE, LINE, PRODUCT
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND         INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND         PRODUCT.P_CODE = LINE.P_CODE
ORDER BY    INVOICE.CUS_CODE, INVOICE.INV_NUMBER, PRODUCT.P_DESCRIPT;
```

30. Modify the query used in Problem 29 to produce the summary shown in Figure P7.30.

FIGURE P7.30 Customer Purchase Summary

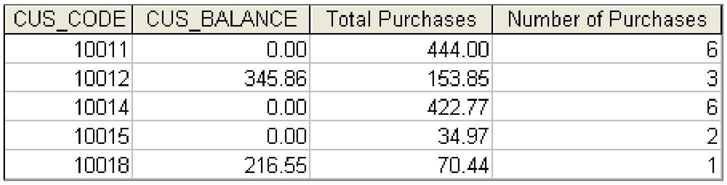


CUS_CODE	CUS_BALANCE	Total Purchases
10011	0.00	444.00
10012	345.86	153.85
10014	0.00	422.77
10015	0.00	34.97
10018	216.55	70.44

```
SELECT     INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE,
           Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Total Purchases]
FROM       CUSTOMER, INVOICE, LINE
WHERE      INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND        CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
GROUP BY   INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE;
```

31. Modify the query in Problem 30 to include the number of individual product purchases made by each customer. (In other words, if the customer's invoice is based on three products, one per LINE_NUMBER, you would count three product purchases. If you examine the original invoice data, you will note that customer 10011 generated three invoices, which contained a total of six lines, each representing a product purchase.) Your output values must match those shown in Figure P7.31.

FIGURE P7.31 Customer Total Purchase Amounts and Number of Purchases

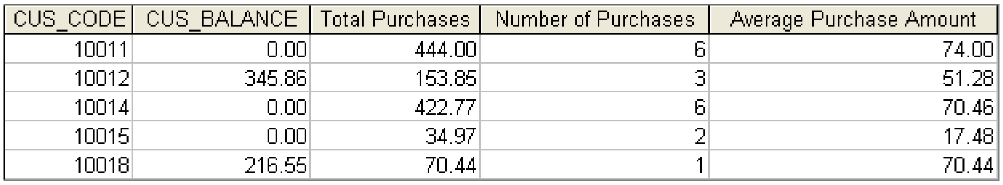


CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases
10011	0.00	444.00	6
10012	345.86	153.85	3
10014	0.00	422.77	6
10015	0.00	34.97	2
10018	216.55	70.44	1

```
SELECT     INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE,
           Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Total Purchases],
           Count(*) AS [Number of Purchases]
FROM       CUSTOMER, INVOICE, LINE
WHERE      INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND        CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
GROUP BY   INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE;
```

32. Use a query to compute the average purchase amount per product made by each customer. (*Hint:* Use the results of Problem 31 as the basis for this query.) Your output values must match those shown in Figure P7.32. Note that the Average Purchase Amount is equal to the Total Purchases divided by the Number of Purchases.

FIGURE P7.32 Average Purchase Amount by Customer

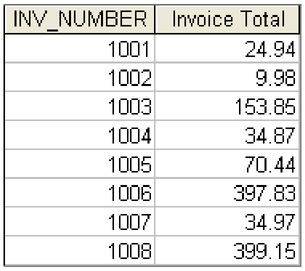


CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases	Average Purchase Amount
10011	0.00	444.00	6	74.00
10012	345.86	153.85	3	51.28
10014	0.00	422.77	6	70.46
10015	0.00	34.97	2	17.48
10018	216.55	70.44	1	70.44

```
SELECT      INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE,
            Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Total Purchases],
            Count(*) AS [Number of Purchases],
            AVG(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Average Purchase Amount]
FROM        CUSTOMER, INVOICE, LINE
WHERE       INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND         CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
GROUP BY    INVOICE.CUS_CODE, CUSTOMER.CUS_BALANCE;
```

33. Create a query to produce the total purchase per invoice, generating the results shown in Figure P7.33. The Invoice Total is the sum of the product purchases in the LINE that corresponds to the INVOICE.

FIGURE P7.33 Invoice Totals

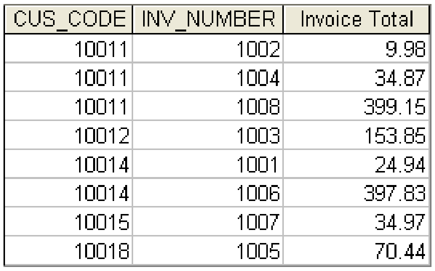


INV_NUMBER	Invoice Total
1001	24.94
1002	9.98
1003	153.85
1004	34.87
1005	70.44
1006	397.83
1007	34.97
1008	399.15

```
SELECT      LINE.INV_NUMBER,
            Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Invoice Total]
FROM        LINE
GROUP BY    LINE.INV_NUMBER;
```

34. Use a query to show the invoices and invoice totals as shown in Figure P7.34. (*Hint*: Group by the CUS_CODE.)

FIGURE P7.34 Invoice Totals by Customer

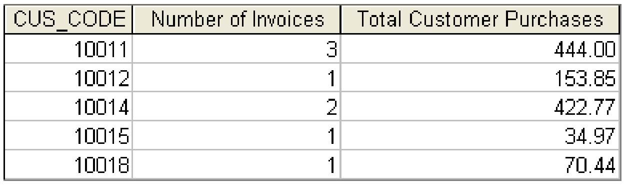


CUS_CODE	INV_NUMBER	Invoice Total
10011	1002	9.98
10011	1004	34.87
10011	1008	399.15
10012	1003	153.85
10014	1001	24.94
10014	1006	397.83
10015	1007	34.97
10018	1005	70.44

```
SELECT          CUS_CODE, LINE.INV_NUMBER AS INV_NUMVER,
                Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Invoice Total]
FROM            INVOICE, LINE
WHERE           INVOICE.INV_NUMBER = LINE.INV_NUMBER
GROUP BY        CUS_CODE, LINE.INV_NUMBER;
```

35. Write a query to produce the number of invoices and the total purchase amounts by customer, using the output shown in Figure P7.35 as your guide. (Compare this summary to the results shown in Problem 34.)

FIGURE P7.35 Number of Invoices and Total Purchase Amounts by Customer



CUS_CODE	Number of Invoices	Total Customer Purchases
10011	3	444.00
10012	1	153.85
10014	2	422.77
10015	1	34.97
10018	1	70.44

Note that a query may be used as the data source for another query. The following code is shown in **qryP7.35A** in your **Ch07_Saleco** database. Note that the data source is **qryP6-34**.

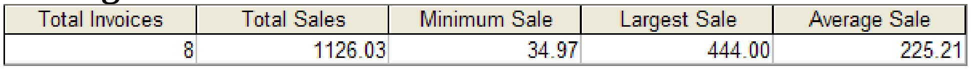
```
SELECT          CUS_CODE,
                Count(INV_NUMBER) AS [Number of Invoices],
                AVG([Invoice Total]) AS [Average Invoice Amount],
                MAX([Invoice Total]) AS [Max  Invoice Amount],
                MIN([Invoice Total]) AS [Min  Invoice Amount],
                Sum([Invoice Total]) AS [Total Customer Purchases]
FROM            [qryP7-34]
GROUP BY        [qryP7-34].CUS_CODE;
```

Instead of using another query as your data source, you can also use an alias. The following code is shown in **Oracle format.** You can also find the MS Access "alias" version in qryP7.35B in your **Ch07_SaleCo** database.)

```
SELECT    CUS_CODE,
          COUNT(LINE.INV_NUMBER) AS [Number of Invoices],
          AVG([Invoice Total]) AS [Average Invoice Amount],
          MAX([Invoice Total]) AS [Max  Invoice Amount],
          MIN([Invoice Total]) AS [Min  Invoice Amount],
          Sum([Invoice Total]) AS [Total Customer Purchases]
FROM      (SELECT     CUS_CODE, LINE.INV_NUMBER AS INV_NUMBER,
                      Sum(LINE.LINE_UNITS*LINE.LINE_PRICE) AS [Invoice Total]
          FROM        INVOICE, LINE
          WHERE       INVOICE.INV_NUMBER = LINE.INV_NUMBER
          GROUP BY CUS_CODE, LINE.INV_NUMBER)
GROUP BY CUS_CODE;
```

36. Using the query results in Problem 35 as your basis, write a query to generate the total number of invoices, the invoice total for all of the invoices, the smallest invoice amount, the largest invoice amount, and the average of all of the invoices. (*Hint*: Check the figure output in Problem 35.) Your output must match Figure P7.36.

FIGURE P7.36 Number of Invoices, Invoice Totals, Minimum, Maximum, and Average Sales

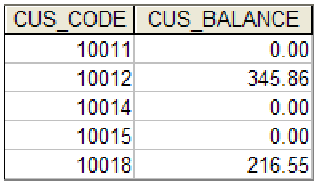


Total Invoices	Total Sales	Minimum Sale	Largest Sale	Average Sale
8	1126.03	34.97	444.00	225.21

```
SELECT   Count([qryP7-34].[INV_NUMBER]) AS [Total Invoices],
         Sum([qryP7-34].[Invoice Total]) AS [Total Sales],
         Min([qryP7-34].[Invoice Total]) AS [Minimum Sale],
         Max([qryP7-34].[Invoice Total]) AS [Largest Sale],
         Avg([qryP7-34].[Invoice Total]) AS [Average Sale]
FROM     [qryP7-34];
```

37. List the balance characteristics of the customers who have made purchases during the current invoice cycle—that is, for the customers who appear in the INVOICE table. The results of this query are shown in Figure P7.37.

FIGURE P7.37 Balances for Customers who Made Purchases



CUS_CODE	CUS_BALANCE
10011	0.00
10012	345.86
10014	0.00
10015	0.00
10018	216.55

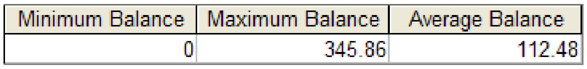
```
SELECT      CUS_CODE, CUS_BALANCE
FROM        CUSTOMER
WHERE       CUSTOMER.CUS_CODE IN
            (SELECT DISTINCT CUS_CODE FROM INVOICE );
```

or

```
SELECT      DISTINCT CUS_CODE, CUS_BALANCE
FROM        CUSTOMER, INVOICE
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE;
```

38. Using the results of the query created in Problem 37, provide a summary of customer balance characteristics as shown in Figure P7.38.

FIGURE P7.38 Balance Summary for Customers Who Made Purchases



Minimum Balance	Maximum Balance	Average Balance
0	345.86	112.48

```
SELECT  MIN(CUS_BALANCE) AS [Minimum Balance],
        MAX(CUS_BALANCE) AS [Maximum Balance],
        AVG(CUS_BALANCE) AS [Average Balance]
FROM    (SELECT     CUS_CODE, CUS_BALANCE
        FROM        CUSTOMER
        WHERE       CUSTOMER.CUS_CODE IN (SELECT DISTINCT CUS_CODE
                                          FROM INVOICE)
        );
```

or

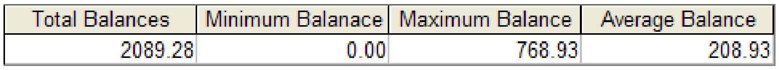
```
SELECT   MIN(CUS_BALANCE) AS [Minimum Balance],
         MAX(CUS_BALANCE) AS [Maximum Balance],
         AVG(CUS_BALANCE) AS [Average Balance]
FROM     (SELECT      DISTINCT CUS_CODE, CUS_BALANCE
         FROM         CUSTOMER, INVOICE
         WHERE        CUSTOMER.CUS_CODE = INVOICE.CUS_CODE);
```

or

```
SELECT   MIN(CUS_BALANCE) AS [Minimum Balance],
         MAX(CUS_BALANCE) AS [Maximum Balance],
         AVG(CUS_BALANCE) AS [Average Balance]
FROM     CUSTOMER
WHERE    CUS_CODE IN (SELECT CUS_CODE FROM INVOICE);
```

39. Create a query to find the customer balance characteristics for all customers, including the total of the outstanding balances. The results of this query are shown in Figure P7.39.

FIGURE P7.39 Customer Balance Summary for All Customers

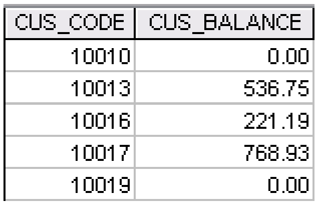


Total Balances	Minimum Balanace	Maximum Balance	Average Balance
2089.28	0.00	768.93	208.93

```
SELECT      Sum(CUS_BALANCE) AS [Total Balance], Min(CUS_BALANCE) AS
            [Minimum Balance], Max(CUS_BALANCE) AS [Maximum Balance],
            Avg(CUS_BALANCE) AS [Average Balance]
FROM        CUSTOMER;
```

40. Find the listing of customers who did not make purchases during the invoicing period. Your output must match the output shown in Figure P7.40.

FIGURE P7.40 Customer Balances for Customers Who Did Not Make Purchases

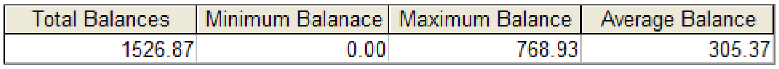


CUS_CODE	CUS_BALANCE
10010	0.00
10013	536.75
10016	221.19
10017	768.93
10019	0.00

```
SELECT   CUS_CODE, CUS_BALANCE
FROM     CUSTOMER
WHERE    CUSTOMER.CUS_CODE NOT IN
         (SELECT DISTINCT CUS_CODE FROM INVOICE);
```

41. Find the customer balance summary for all customers who have not made purchases during the current invoicing period. The results are shown in Figure P7.41.

FIGURE P7.41 Summary of Customer Balances for Customers Who Did Not Make Purchases



Total Balances	Minimum Balanace	Maximum Balance	Average Balance
1526.87	0.00	768.93	305.37

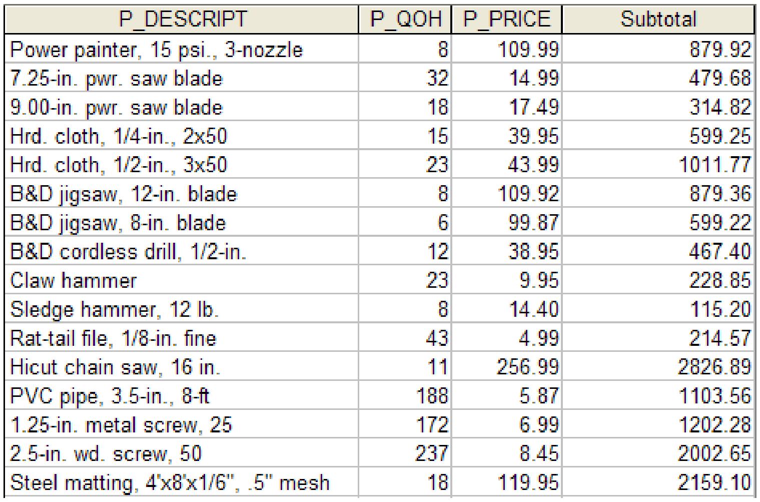
```
SELECT      SUM(CUS_BALANCE) AS [Total Balance],
            MIN(CUS_BALANCE) AS [Minimum Balance],
            MAX(CUS_BALANCE) AS [Maximum Balance],
            AVG(CUS_BALANCE) AS [Average Balance]
FROM        (SELECT     CUS_CODE, CUS_BALANCE
            FROM        CUSTOMER
           WHERE        CUSTOMER.CUS_CODE NOT IN
                        (SELECT DISTINCT CUS_CODE FROM INVOICE)
           );
```

or

```
SELECT   SUM(CUS_BALANCE) AS [Total Balance],
         MIN(CUS_BALANCE) AS [Minimum Balance],
         MAX(CUS_BALANCE) AS [Maximum Balance],
         AVG(CUS_BALANCE) AS [Average Balance]
FROM     CUSTOMER
WHERE    CUS_CODE NOT IN (SELECT CUS_CODE FROM INVOICE);
```

42. Create a query to produce the summary of the value of products currently in inventory. Note that the value of each product is produced by the multiplication of the units currently in inventory and the unit price. Use the ORDER BY clause to match the order shown in Figure P7.42.

FIGURE P7.42 Value of Products in Inventory

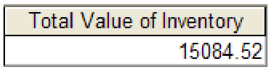


P_DESCRIPT	P_QOH	P_PRICE	Subtotal
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT      P_DESCRIPT, P_QOH, P_PRICE, P_QOH*P_PRICE AS Subtotal
FROM        PRODUCT;
```

43. Using the results of the query created in Problem 42, find the total value of the product inventory. The results are shown in Figure P7.43.

FIGURE P7.43 Total Value of All Products in Inventory



Total Value of Inventory
15084.52

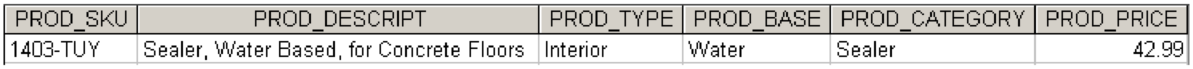
```
SELECT      SUM(P_QOH*P_PRICE) AS [Total Value of Inventory]
FROM        PRODUCT;
```

44. Write a query to display the eight departments in the LGDEPARTMENT table.

```
SELECT *
FROM LGDEPARTMENT;
```

45. Write a query to display the SKU (stock keeping unit), description, type, base, category, and price for all products that have a PROD_BASE of water and a PROD_CATEGORY of sealer.

FIGURE P7. 45 WATER-BASED SEALERS

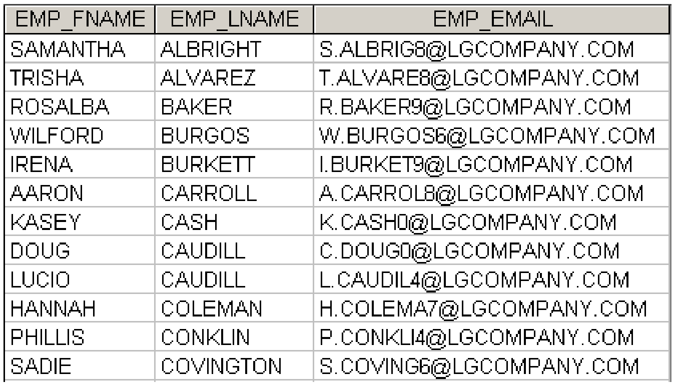


PROD_SKU	PROD_DESCRIPT	PROD_TYPE	PROD_BASE	PROD_CATEGORY	PROD_PRICE
1403-TUY	Sealer, Water Based, for Concrete Floors	Interior	Water	Sealer	42.99

```
SELECT PROD_SKU, PROD_DESCRIPT, PROD_TYPE, PROD_BASE, PROD_CATEGORY,
            PROD_PRICE
FROM LGPRODUCT
WHERE PROD_BASE='Water' And PROD_CATEGORY='Sealer';
```

46. Write a query to display the first name, last name, and e-mail address of employees hired from January 1, 2001, to December 31, 2010. Sort the output by last name and then by first name.

FIGURE P7. 46 Employees hired from 2001–2010

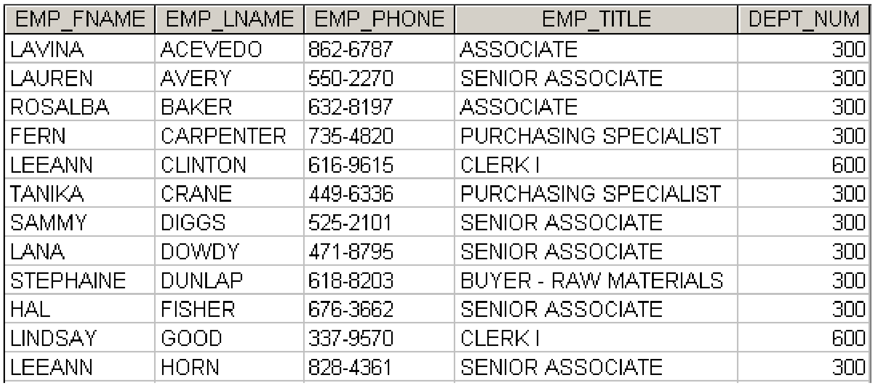


EMP_FNAME	EMP_LNAME	EMP_EMAIL
SAMANTHA	ALBRIGHT	S.ALBRIG8@LGCOMPANY.COM
TRISHA	ALVAREZ	T.ALVARE8@LGCOMPANY.COM
ROSALBA	BAKER	R.BAKER9@LGCOMPANY.COM
WILFORD	BURGOS	W.BURGOS6@LGCOMPANY.COM
IRENA	BURKETT	I.BURKET9@LGCOMPANY.COM
AARON	CARROLL	A.CARROL8@LGCOMPANY.COM
KASEY	CASH	K.CASH0@LGCOMPANY.COM
DOUG	CAUDILL	C.DOUG0@LGCOMPANY.COM
LUCIO	CAUDILL	L.CAUDIL4@LGCOMPANY.COM
HANNAH	COLEMAN	H.COLEMA7@LGCOMPANY.COM
PHILLIS	CONKLIN	P.CONKLI4@LGCOMPANY.COM
SADIE	COVINGTON	S.COVING6@LGCOMPANY.COM

```
SELECT EMP_FNAME, EMP_LNAME, EMP_EMAIL
FROM LGEMPLOYEE
WHERE EMP_HIREDATE Between ‘1/1/2001’ And ‘12/31/2010’
ORDER BY EMP_LNAME, EMP_FNAME;
```

47. Write a query to display the first name, last name, phone number, title, and department number of employees who work in department 300 or have the title "CLERK I." Sort the output by last name and then by first name.

FIGURE P7. 47 Clerks and employees in department 300

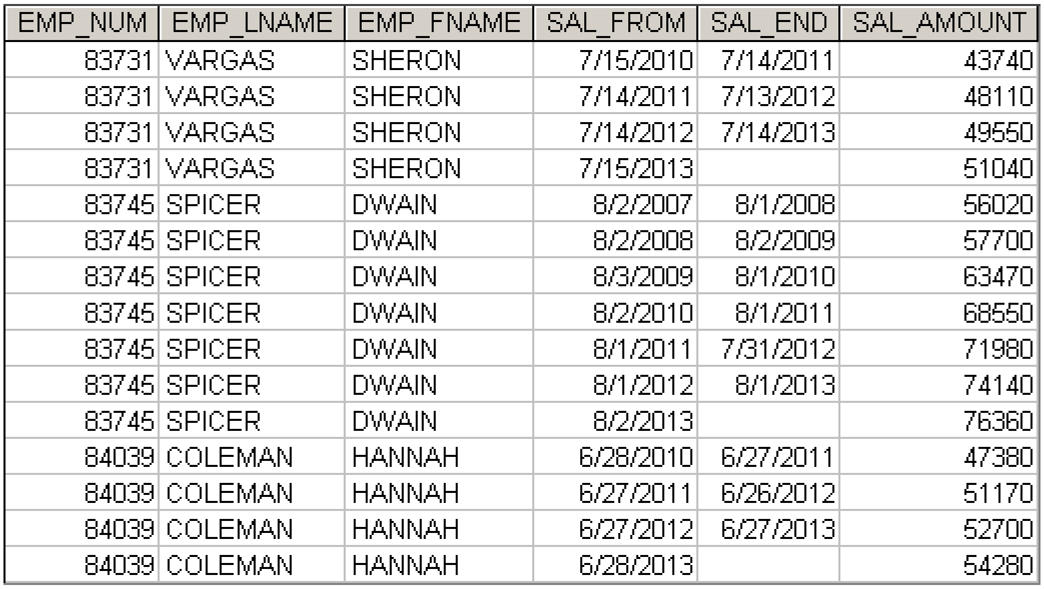


EMP_FNAME	EMP_LNAME	EMP_PHONE	EMP_TITLE	DEPT_NUM
LAVINA	ACEVEDO	862-6787	ASSOCIATE	300
LAUREN	AVERY	550-2270	SENIOR ASSOCIATE	300
ROSALBA	BAKER	632-8197	ASSOCIATE	300
FERN	CARPENTER	735-4820	PURCHASING SPECIALIST	300
LEEANN	CLINTON	616-9615	CLERK I	600
TANIKA	CRANE	449-6336	PURCHASING SPECIALIST	300
SAMMY	DIGGS	525-2101	SENIOR ASSOCIATE	300
LANA	DOWDY	471-8795	SENIOR ASSOCIATE	300
STEPHAINE	DUNLAP	618-8203	BUYER - RAW MATERIALS	300
HAL	FISHER	676-3662	SENIOR ASSOCIATE	300
LINDSAY	GOOD	337-9570	CLERK I	600
LEEANN	HORN	828-4361	SENIOR ASSOCIATE	300

```
SELECT EMP_FNAME, EMP_LNAME, EMP_PHONE, EMP_TITLE, DEPT_NUM
FROM LGEMPLOYEE
WHERE DEPT_NUM=300 Or EMP_TITLE='CLERK I'
ORDER BY EMP_LNAME, EMP_FNAME;
```

48. Write a query to display the employee number, last name, first name, salary "from" date, salary end date, and salary amount for employees 83731, 83745, and 84039. Sort the output by employee number and salary "from" date.

FIGURE P7. 48 Salary history for selected employees



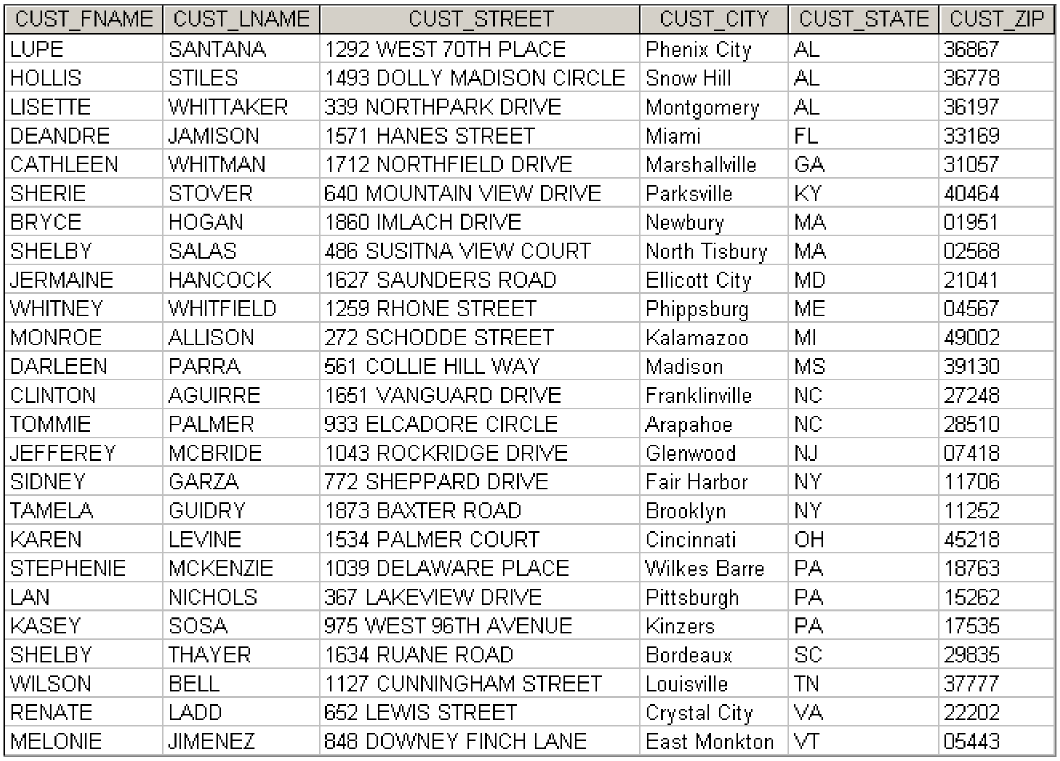
EMP_NUM	EMP_LNAME	EMP_FNAME	SAL_FROM	SAL_END	SAL_AMOUNT
83731	VARGAS	SHERON	7/15/2010	7/14/2011	43740
83731	VARGAS	SHERON	7/14/2011	7/13/2012	48110
83731	VARGAS	SHERON	7/14/2012	7/14/2013	49550
83731	VARGAS	SHERON	7/15/2013		51040
83745	SPICER	DWAIN	8/2/2007	8/1/2008	56020
83745	SPICER	DWAIN	8/2/2008	8/2/2009	57700
83745	SPICER	DWAIN	8/3/2009	8/1/2010	63470
83745	SPICER	DWAIN	8/2/2010	8/1/2011	68550
83745	SPICER	DWAIN	8/1/2011	7/31/2012	71980
83745	SPICER	DWAIN	8/1/2012	8/1/2013	74140
83745	SPICER	DWAIN	8/2/2013		76360
84039	COLEMAN	HANNAH	6/28/2010	6/27/2011	47380
84039	COLEMAN	HANNAH	6/27/2011	6/26/2012	51170
84039	COLEMAN	HANNAH	6/27/2012	6/27/2013	52700
84039	COLEMAN	HANNAH	6/28/2013		54280

```
SELECT EMP.EMP_NUM, EMP_LNAME, EMP_FNAME, SAL_FROM, SAL_END,
SAL_AMOUNT
FROM LGEMPLOYEE AS EMP, LGSALARY_HISTORY AS SAL
```

```
WHERE EMP.EMP_NUM=SAL.EMP_NUM And EMP.EMP_NUM In (83731,83745,84039)
ORDER BY EMP.EMP_NUM, SAL_FROM;
```

49. Write a query to display the first name, last name, street, city, state, and zip code of any customer who purchased a Foresters Best brand top coat between July 15, 2013, and July 31, 2013. If a customer purchased more than one such product, display the customer's information only once in the output. Sort the output by state, last name, and then first name.

FIGURE P7. 49 Customers who purchased Foresters Best top coat

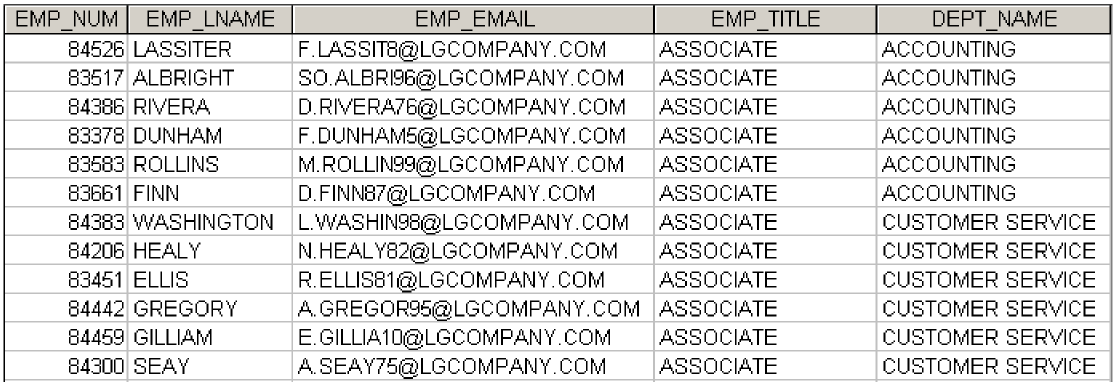


CUST_FNAME	CUST_LNAME	CUST_STREET	CUST_CITY	CUST_STATE	CUST_ZIP
LUPE	SANTANA	1292 WEST 70TH PLACE	Phenix City	AL	36867
HOLLIS	STILES	1493 DOLLY MADISON CIRCLE	Snow Hill	AL	36778
LISETTE	WHITTAKER	339 NORTHPARK DRIVE	Montgomery	AL	36197
DEANDRE	JAMISON	1571 HANES STREET	Miami	FL	33169
CATHLEEN	WHITMAN	1712 NORTHFIELD DRIVE	Marshallville	GA	31057
SHERIE	STOVER	640 MOUNTAIN VIEW DRIVE	Parksville	KY	40464
BRYCE	HOGAN	1860 IMLACH DRIVE	Newbury	MA	01951
SHELBY	SALAS	486 SUSITNA VIEW COURT	North Tisbury	MA	02568
JERMAINE	HANCOCK	1627 SAUNDERS ROAD	Ellicott City	MD	21041
WHITNEY	WHITFIELD	1259 RHONE STREET	Phippsburg	ME	04567
MONROE	ALLISON	272 SCHODDE STREET	Kalamazoo	MI	49002
DARLEEN	PARRA	561 COLLIE HILL WAY	Madison	MS	39130
CLINTON	AGUIRRE	1651 VANGUARD DRIVE	Franklinville	NC	27248
TOMMIE	PALMER	933 ELCADORE CIRCLE	Arapahoe	NC	28510
JEFFEREY	MCBRIDE	1043 ROCKRIDGE DRIVE	Glenwood	NJ	07418
SIDNEY	GARZA	772 SHEPPARD DRIVE	Fair Harbor	NY	11706
TAMELA	GUIDRY	1873 BAXTER ROAD	Brooklyn	NY	11252
KAREN	LEVINE	1534 PALMER COURT	Cincinnati	OH	45218
STEPHENIE	MCKENZIE	1039 DELAWARE PLACE	Wilkes Barre	PA	18763
LAN	NICHOLS	367 LAKEVIEW DRIVE	Pittsburgh	PA	15262
KASEY	SOSA	975 WEST 96TH AVENUE	Kinzers	PA	17535
SHELBY	THAYER	1634 RUANE ROAD	Bordeaux	SC	29835
WILSON	BELL	1127 CUNNINGHAM STREET	Louisville	TN	37777
RENATE	LADD	652 LEWIS STREET	Crystal City	VA	22202
MELONIE	JIMENEZ	848 DOWNEY FINCH LANE	East Monkton	VT	05443

```
SELECT DISTINCT CUST_FNAME, CUST_LNAME, CUST_STREET, CUST_CITY,
CUST_STATE, CUST_ZIP
FROM LGCUSTOMER AS C, LGINVOICE AS I, LGLINE AS L, LGPRODUCT AS P,
LGBRAND AS B
WHERE C.CUST_CODE = I.CUST_CODE
AND   I.INV_NUM = L.INV_NUM
AND   L.PROD_SKU = P.PROD_SKU
AND   P.BRAND_ID = B.BRAND_ID
AND   BRAND_NAME = ‘FORESTERS BEST’
AND   PROD_CATEGORY = ‘Top Coat’
AND   INV_DATE BETWEEN ‘15-JUL-2013’ AND ‘31-JUL-2013’
ORDER BY CUST_STATE, CUST_LNAME, CUST_FNAME;
```

50. Write a query to display the employee number, last name, e-mail address, title, and department name of each employee whose job title ends in the word "ASSOCIATE." Sort the output by department name and employee title.

FIGURE P7. 50 Employees with the Associate title

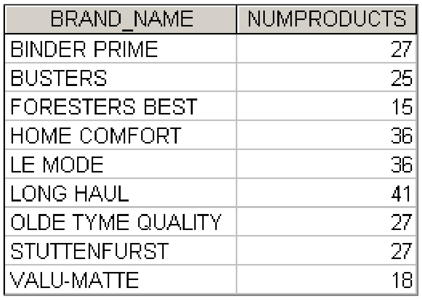


EMP_NUM	EMP_LNAME	EMP_EMAIL	EMP_TITLE	DEPT_NAME
84526	LASSITER	F.LASSIT8@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
83517	ALBRIGHT	SO.ALBRI96@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
84386	RIVERA	D.RIVERA76@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
83378	DUNHAM	F.DUNHAM5@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
83583	ROLLINS	M.ROLLIN99@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
83661	FINN	D.FINN87@LGCOMPANY.COM	ASSOCIATE	ACCOUNTING
84383	WASHINGTON	L.WASHIN98@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE
84206	HEALY	N.HEALY82@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE
83451	ELLIS	R.ELLIS81@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE
84442	GREGORY	A.GREGOR95@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE
84459	GILLIAM	E.GILLIA10@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE
84300	SEAY	A.SEAY75@LGCOMPANY.COM	ASSOCIATE	CUSTOMER SERVICE

```
SELECT E.EMP_NUM, EMP_LNAME, EMP_EMAIL, EMP_TITLE, DEPT_NAME
FROM LGEMPLOYEE AS E, LGDEPARTMENT AS D
WHERE E.DEPT_NUM = D.DEPT_NUM
AND   EMP_TITLE LIKE '%ASSOCIATE'
ORDER BY DEPT_NAME, EMP_TITLE;
```

51. Write a query to display a brand name and the number of products of that brand that are in the database. Sort the output by the brand name.

FIGURE P7. 51 Number of products of each brand

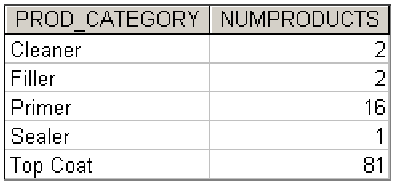


BRAND_NAME	NUMPRODUCTS
BINDER PRIME	27
BUSTERS	25
FORESTERS BEST	15
HOME COMFORT	36
LE MODE	36
LONG HAUL	41
OLDE TYME QUALITY	27
STUTTENFURST	27
VALU-MATTE	18

```
SELECT BRAND_NAME, Count(PROD_SKU) AS NUMPRODUCTS
FROM LGBRAND AS B, LGPRODUCT AS P
WHERE B.BRAND_ID = P.BRAND_ID
GROUP BY BRAND_NAME
ORDER BY BRAND_NAME;
```

52. Write a query to display the number of products in each category that have a water base.

FIGURE P7. 52 Number of water-based products in each category

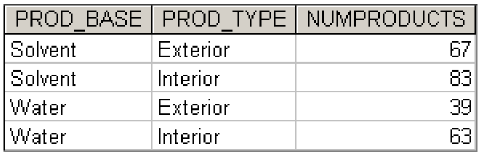


PROD_CATEGORY	NUMPRODUCTS
Cleaner	2
Filler	2
Primer	16
Sealer	1
Top Coat	81

```
SELECT PROD_CATEGORY, Count(*) AS NUMPRODUCTS
FROM LGPRODUCT
WHERE PROD_BASE = 'Water'
GROUP BY PROD_CATEGORY;
```

53. Write a query to display the number of products within each base and type combination.

FIGURE P7. 53 Number of products of each base and type

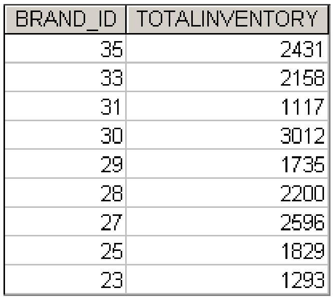


PROD_BASE	PROD_TYPE	NUMPRODUCTS
Solvent	Exterior	67
Solvent	Interior	83
Water	Exterior	39
Water	Interior	63

```
SELECT PROD_BASE, PROD_TYPE, Count(*) AS NUMPRODUCTS
FROM LGPRODUCT
GROUP BY PROD_BASE, PROD_TYPE
ORDER BY PROD_BASE, PROD_TYPE;
```

54. Write a query to display the total inventory—that is, the sum of all products on hand for each brand ID. Sort the output by brand ID in descending order.

FIGURE P7. 54 Total inventory of each brand of products

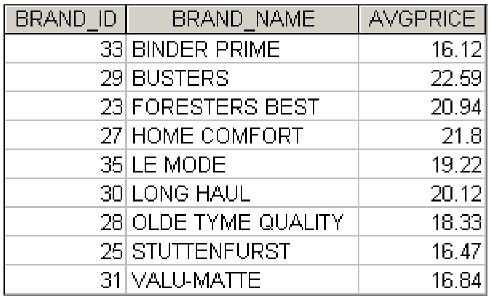


BRAND_ID	TOTALINVENTORY
35	2431
33	2158
31	1117
30	3012
29	1735
28	2200
27	2596
25	1829
23	1293

```
SELECT BRAND_ID, Sum(PROD_QOH) AS TOTALINVENTORY
FROM LGPRODUCT
GROUP BY BRAND_ID
ORDER BY BRAND_ID DESC;
```

55. Write a query to display the brand ID, brand name, and average price of products of each brand. Sort the output by brand name. (Results are shown with the average price rounded to two decimal places.)

FIGURE P7. 55 Average price of products of each brand

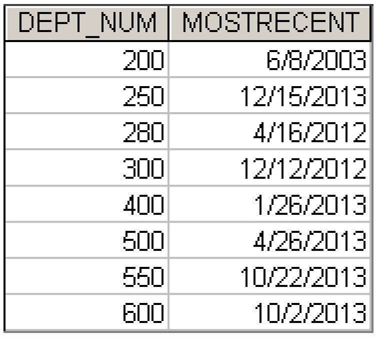


BRAND_ID	BRAND_NAME	AVGPRICE
33	BINDER PRIME	16.12
29	BUSTERS	22.59
23	FORESTERS BEST	20.94
27	HOME COMFORT	21.8
35	LE MODE	19.22
30	LONG HAUL	20.12
28	OLDE TYME QUALITY	18.33
25	STUTTENFURST	16.47
31	VALU-MATTE	16.84

```
SELECT P.BRAND_ID, BRAND_NAME, Round(Avg(PROD_PRICE),2) AS AVGPRICE
FROM LGBRAND AS B, LGPRODUCT AS P
WHERE B.BRAND_ID = P.BRAND_ID
GROUP BY P.BRAND_ID, BRAND_NAME
ORDER BY BRAND_NAME;
```

56. Write a query to display the department number and most recent employee hire date for each department. Sort the output by department number.

FIGURE P7. 56 Most recent hire in each department

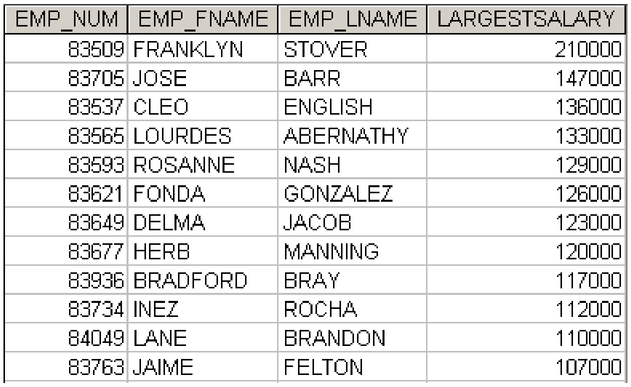


DEPT_NUM	MOSTRECENT
200	6/8/2003
250	12/15/2013
280	4/16/2012
300	12/12/2012
400	1/26/2013
500	4/26/2013
550	10/22/2013
600	10/2/2013

```
SELECT DEPT_NUM, Max(EMP_HIREDATE) AS MOSTRECENT
FROM LGEMPLOYEE
GROUP BY DEPT_NUM
ORDER BY DEPT_NUM;
```

57. Write a query to display the employee number, first name, last name, and largest salary amount for each employee in department 200. Sort the output by largest salary in descending order.

FIGURE P7. 57 Largest salary amount for each employee in department 200

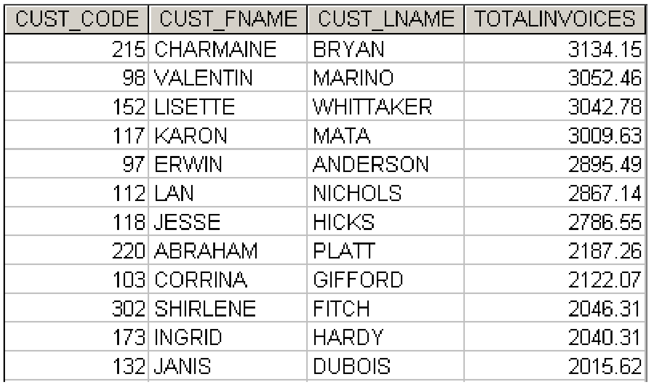


EMP_NUM	EMP_FNAME	EMP_LNAME	LARGESTSALARY
83509	FRANKLYN	STOVER	210000
83705	JOSE	BARR	147000
83537	CLEO	ENGLISH	136000
83565	LOURDES	ABERNATHY	133000
83593	ROSANNE	NASH	129000
83621	FONDA	GONZALEZ	126000
83649	DELMA	JACOB	123000
83677	HERB	MANNING	120000
83936	BRADFORD	BRAY	117000
83734	INEZ	ROCHA	112000
84049	LANE	BRANDON	110000
83763	JAIME	FELTON	107000

```
SELECT E.EMP_NUM, EMP_FNAME, EMP_LNAME, Max(SAL_AMOUNT) AS
LARGESTSALARY
FROM LGEMPLOYEE AS E, LGSALARY_HISTORY AS S
WHERE E.EMP_NUM = S.EMP_NUM
AND DEPT_NUM = 200
GROUP BY E.EMP_NUM, EMP_FNAME, EMP_LNAME
ORDER BY max(sal_amount) DESC;
```

58. Write a query to display the customer code, first name, last name, and sum of all invoice totals for customers with cumulative invoice totals greater than $1,500. Sort the output by the sum of invoice totals in descending order.

FIGURE P7. 58 List of customers with cumulative purchases of more than $1,500



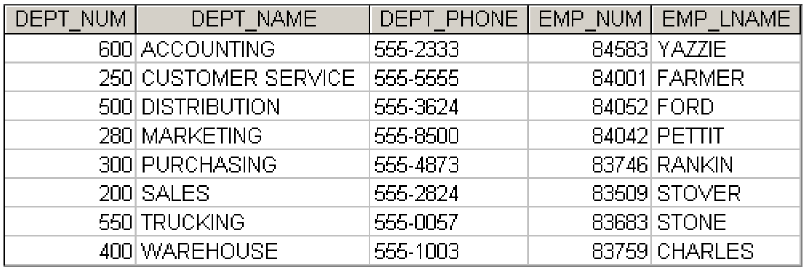
CUST_CODE	CUST_FNAME	CUST_LNAME	TOTALINVOICES
215	CHARMAINE	BRYAN	3134.15
98	VALENTIN	MARINO	3052.46
152	LISETTE	WHITTAKER	3042.78
117	KARON	MATA	3009.63
97	ERWIN	ANDERSON	2895.49
112	LAN	NICHOLS	2867.14
118	JESSE	HICKS	2786.55
220	ABRAHAM	PLATT	2187.26
103	CORRINA	GIFFORD	2122.07
302	SHIRLENE	FITCH	2046.31
173	INGRID	HARDY	2040.31
132	JANIS	DUBOIS	2015.62

```
SELECT C.CUST_CODE, CUST_FNAME, CUST_LNAME, Sum(INV_TOTAL) AS
TOTALINVOICES
FROM LGCUSTOMER AS C, LGINVOICE AS I
WHERE C.CUST_CODE = I.CUST_CODE
GROUP BY C.CUST_CODE, CUST_FNAME, CUST_LNAME
```

```
HAVING Sum(INV_TOTAL) > 1500
ORDER BY Sum(INV_TOTAL) DESC;
```

59. Write a query to display the department number, department name, department phone number, employee number, and last name of each department manager. Sort the output by department name.

FIGURE P7. 59 Department managers

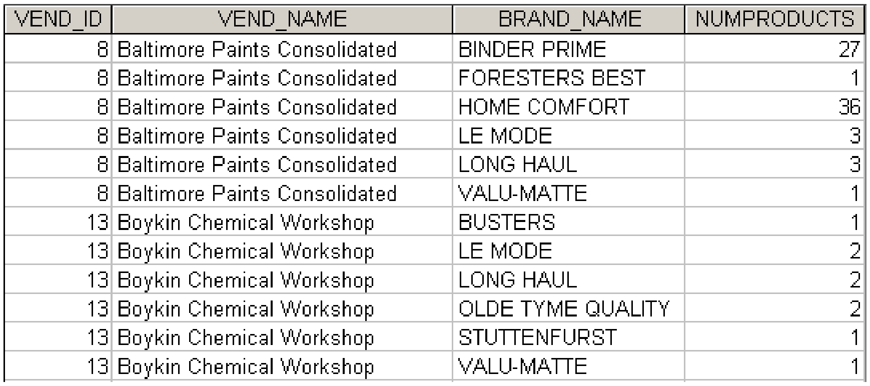


DEPT_NUM	DEPT_NAME	DEPT_PHONE	EMP_NUM	EMP_LNAME
600	ACCOUNTING	555-2333	84583	YAZZIE
250	CUSTOMER SERVICE	555-5555	84001	FARMER
500	DISTRIBUTION	555-3624	84052	FORD
280	MARKETING	555-8500	84042	PETTIT
300	PURCHASING	555-4873	83746	RANKIN
200	SALES	555-2824	83509	STOVER
550	TRUCKING	555-0057	83683	STONE
400	WAREHOUSE	555-1003	83759	CHARLES

```
SELECT D.DEPT_NUM, DEPT_NAME, DEPT_PHONE, D.EMP_NUM, EMP_LNAME
FROM LGDEPARTMENT AS D, LGEMPLOYEE AS E
WHERE D.EMP_NUM = E.EMP_NUM
ORDER BY DEPT_NAME;
```

60. Write a query to display the vendor ID, vendor name, brand name, and number of products of each brand supplied by each vendor. Sort the output by vendor name and then by brand name.

FIGURE P7. 60 Number of products of each brand supplied by each vendor

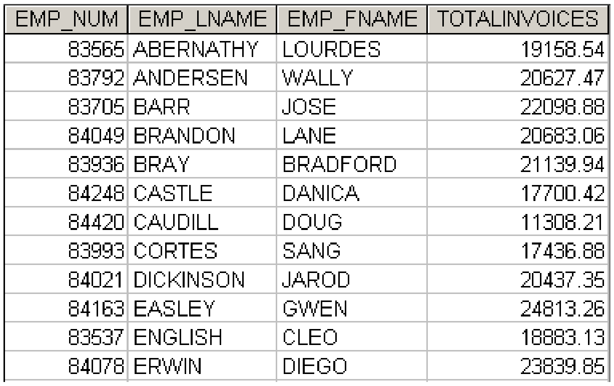


VEND_ID	VEND_NAME	BRAND_NAME	NUMPRODUCTS
8	Baltimore Paints Consolidated	BINDER PRIME	27
8	Baltimore Paints Consolidated	FORESTERS BEST	1
8	Baltimore Paints Consolidated	HOME COMFORT	36
8	Baltimore Paints Consolidated	LE MODE	3
8	Baltimore Paints Consolidated	LONG HAUL	3
8	Baltimore Paints Consolidated	VALU-MATTE	1
13	Boykin Chemical Workshop	BUSTERS	1
13	Boykin Chemical Workshop	LE MODE	2
13	Boykin Chemical Workshop	LONG HAUL	2
13	Boykin Chemical Workshop	OLDE TYME QUALITY	2
13	Boykin Chemical Workshop	STUTTENFURST	1
13	Boykin Chemical Workshop	VALU-MATTE	1

```
SELECT V.VEND_ID, VEND_NAME, BRAND_NAME, Count(*) AS NUMPRODUCTS
FROM LGBRAND AS B, LGPRODUCT AS P, LGSUPPLIES AS S, LGVENDOR AS V
WHERE B.BRAND_ID = P.BRAND_ID
AND P.PROD_SKU = S.PROD_SKU
AND S.VEND_ID = V.VEND_ID
GROUP BY V.VEND_ID, VEND_NAME, BRAND_NAME
ORDER BY VEND_NAME, BRAND_NAME;
```

61. Write a query to display the employee number, last name, first name, and sum of invoice totals for all employees who completed an invoice. Sort the output by employee last name and then by first name.

FIGURE P7. 61 Total value of invoices completed by each employee

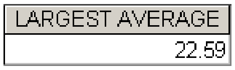


EMP_NUM	EMP_LNAME	EMP_FNAME	TOTALINVOICES
83565	ABERNATHY	LOURDES	19158.54
83792	ANDERSEN	WALLY	20627.47
83705	BARR	JOSE	22098.88
84049	BRANDON	LANE	20683.06
83936	BRAY	BRADFORD	21139.94
84248	CASTLE	DANICA	17700.42
84420	CAUDILL	DOUG	11308.21
83993	CORTES	SANG	17436.88
84021	DICKINSON	JAROD	20437.35
84163	EASLEY	GWEN	24813.26
83537	ENGLISH	CLEO	18883.13
84078	ERWIN	DIEGO	23839.85

```
SELECT EMP_NUM, EMP_LNAME, EMP_FNAME, Sum(INV_TOTAL) AS TOTALINVOICES
FROM LGINVOICE, LGEMPLOYEE
WHERE EMP_NUM = EMPLOYEE_ID
GROUP BY EMP_NUM, EMP_LNAME, EMP_FNAME
ORDER BY EMP_LNAME, EMP_FNAME;
```

62. Write a query to display the largest average product price of any brand.

FIGURE P7. 62 Largest average brand price

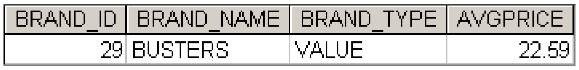


LARGEST AVERAGE
22.59

```
SELECT Max(AVGPRICE) AS “LARGEST AVERAGE”
FROM (SELECT BRAND_ID, Round(Avg(PROD_PRICE),2) AS AVGPRICE
    FROM LGPRODUCT P
    GROUP BY BRAND_ID);
```

63. Write a query to display the brand ID, brand name, brand type, and average price of products for the brand that has the largest average product price.

FIGURE P7. 63 Brand with highest average price



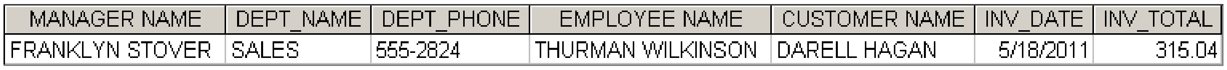
BRAND_ID	BRAND_NAME	BRAND_TYPE	AVGPRICE
29	BUSTERS	VALUE	22.59

```
SELECT P.BRAND_ID, BRAND_NAME, BRAND_TYPE,
           Round(Avg(PROD_PRICE),2) AS AVGPRICE
FROM LGPRODUCT AS P, LGBRAND AS B
```

```
WHERE P.BRAND_ID = B.BRAND_ID
GROUP BY P.BRAND_ID, BRAND_NAME, BRAND_TYPE
HAVING Round(Avg(PROD_PRICE),2) =
                    (SELECT Max(AVGPRICE) AS "LARGEST AVERAGE"
                      FROM (SELECT BRAND_ID, Round(Avg(PROD_PRICE),2) AS AVGPRICE
                                FROM LGPRODUCT P
                                GROUP BY BRAND_ID));
```

64. Write a query to display the manager name, department name, department phone number, employee name, customer name, invoice date, and invoice total for the department manager of the employee who made a sale to a customer whose last name is Hagan on May 18, 2011. For all person names, concatenate the first and last names into a single field.

FIGURE P7. 64 Manager of employee making a sale to customer Hagan



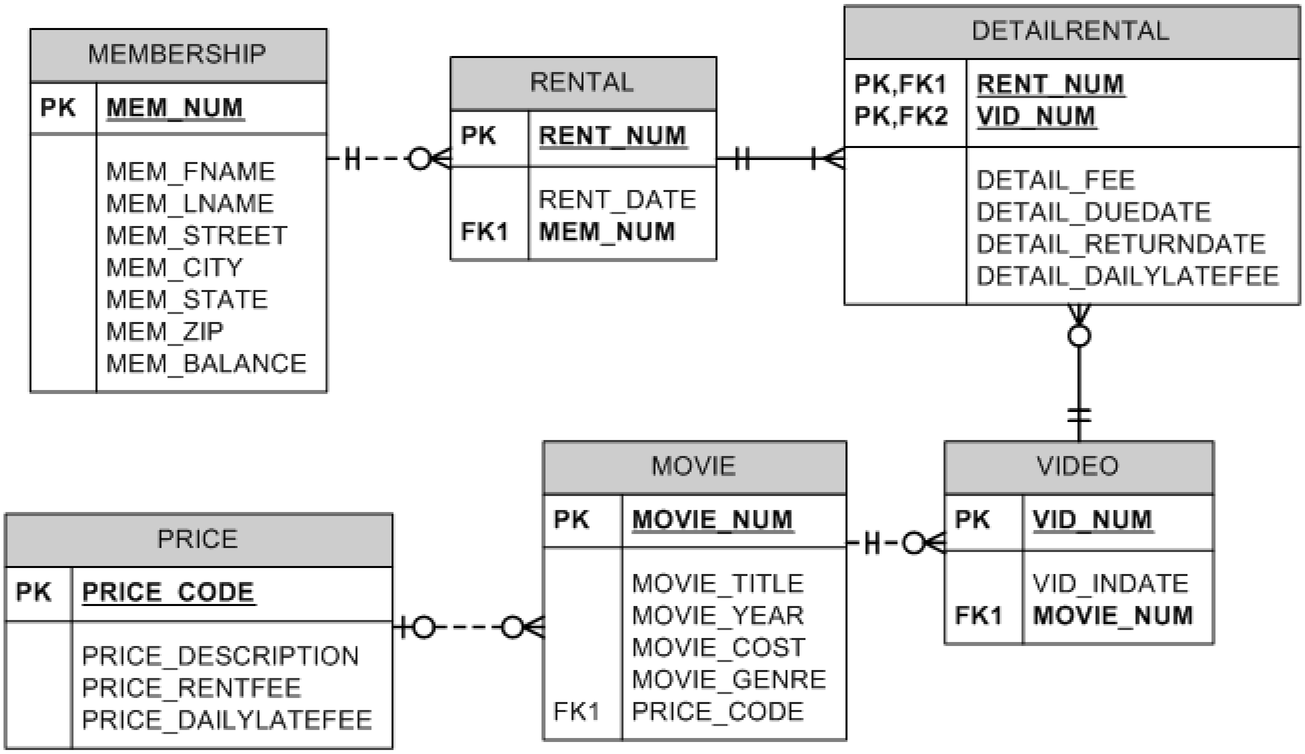
MANAGER NAME	DEPT_NAME	DEPT_PHONE	EMPLOYEE NAME	CUSTOMER NAME	INV_DATE	INV_TOTAL
FRANKLYN STOVER	SALES	555-2824	THURMAN WILKINSON	DARELL HAGAN	5/18/2011	315.04

```
SELECT DE.EMP_FNAME || ' ' || DE.EMP_LNAME AS “MANAGER NAME”, DEPT_NAME,
DEPT_PHONE, E.EMP_FNAME & ' ' & E.EMP_LNAME AS “EMPLOYEE NAME”,
CUST_FNAME || ' ' || CUST_LNAME AS “CUSTOMER NAME”, INV_DATE, INV_TOTAL
FROM LGDEPARTMENT AS D, LGEMPLOYEE AS E, LGEMPLOYEE AS DE, LGINVOICE
AS I, LGCUSTOMER AS C
WHERE D.EMP_NUM = DE.EMP_NUM
AND D.DEPT_NUM = E.DEPT_NUM
AND E.EMP_NUM = I.EMPLOYEE_ID
AND I.CUST_CODE = C.CUST_CODE
AND CUST_LNAME = 'HAGAN'
AND INV_DATE = ‘18-MAY-11’;
```

CASES

EliteVideo is a startup company providing concierge DVD kiosk service in upscale neighborhoods. EliteVideo can own several copies (VIDEO) of each movie (MOVIE). For example, the store may have 10 copies of the movie “Twist in the Wind”. “Twist in the Wind” would be one MOVIE and each copy would be a VIDEO. A rental transaction (RENTAL) involves one or more videos being rented to a member (MEMBERSHIP). A video can be rented many times over its lifetime, therefore, there is a M:N relationship between RENTAL and VIDEO. DETAILRENTAL is the bridge table to resolve this relationship. The complete ERD is provided in Figure P7.65.

Figure P7.65 EliteVideo ERD



65. Write the SQL code to create the table structures for the entities shown in Figure P7.65. The structures should contain the attributes specified in the ERD. Use data types that would be appropriate for the data that will need to be stored in each attribute. Enforce primary key and foreign key constraints as indicated by the ERD.

Based on the referential integrity constraints, students should be able to identify a correct sequence in which to create the tables. The key point is that due to referential integrity constraints, the table contributing its PK as a FK must be created before the related table containing the FK.

```
CREATE TABLE PRICE (
PRICE_CODE              NUMBER(2,0)  PRIMARY KEY,
PRICE_DESCRIPTION       VARCHAR2(20)  NOT NULL  ,
PRICE_RENTFEE           NUMBER(5,2) CHECK (PRICE_RENTFEE >= 0),
PRICE_DAILYLATEFEE      NUMBER(5,2) CHECK (PRICE_DAILYLATEFEE >= 0)
);

CREATE TABLE MOVIE (
MOVIE_NUM          NUMBER(8,0)  PRIMARY KEY,
MOVIE_TITLE        VARCHAR2(75) NOT NULL,
MOVIE_YEAR         NUMBER(4,0)  CHECK (MOVIE_YEAR > 1900),
MOIVE_COST         NUMBER(5,2),
MOVIE_GENRE        VARCHAR2(50),
PRICE_CODE         NUMBER(2,0) CONSTRAINT MOVIE_PRICE_CODE_FK
REFERENCES PRICE
);

CREATE TABLE VIDEO (
VID_NUM             NUMBER(8,0) PRIMARY KEY,
VID_INDATE          DATE,
MOVIE_NUM           NUMBER(8,0) CONSTRAINT VIDEO_MOVIE_NUM_FK
REFERENCES MOVIE
);

CREATE TABLE MEMBERSHIP (
MEM_NUM             NUMBER(8,0) PRIMARY KEY,
MEM_FNAME           VARCHAR2(30) NOT NULL,
MEM_LNAME           VARCHAR2(30) NOT NULL,
MEM_STREET          VARCHAR2(120),
MEM_CITY            VARCHAR2(50),
MEM_STATE           CHAR(2),
MEM_ZIP             CHAR(5)
MEM_BALANCE         NUMBER(10,2)
);

CREATE TABLE RENTAL (
RENT_NUM  NUMBER(8,0),
RENT_DATE DATE DEFAULT SYSDATE,
MEM_NUM   NUMBER(8,0) CONSTRAINT RENTAL_MEM_NUM_FK REFERENCES
MEMBERSHIP
);

CREATE TABLE DETAILRENTAL (
RENT_NUM                NUMBER(8,0) CONSTRAINT DETAIL_RENT_NUM_FK
REFERENCES RENTAL,
VID_NUM                 NUMBER(8,0) CONSTRAINT DETAIL_VID_NUM_FK
REFERENCE VIDEO,
DETAIL_FEE              NUMBER(5,2),
DETAIL_DUEDATE          DATE,
DETAIL_RETURNDATE       DATE,
DETAIL_DAILYLATEFEE     NUMBER(5,2),
CONSTRAINT DETAIL_RENT_VID_PK PRIMARY KEY (RENT_NUM, VID_NUM)
);
```

66. The following tables provide a very small portion of the data that will be kept in the database. This data needs to be inserted into the database for testing purposes. Write the INSERT commands necessary to place the following data in the tables that were created in problem 65.

MEMBERSHIP

Mem_Num	Mem_Fname	Mem_Lname	Mem_Street	Mem_City	Mem_State	Mem_Zip	Mem_Balance
102	Tami	Dawson	2632 Takli Circle	Norene	TN	37136	11
103	Curt	Knight	4025 Cornell Court	Flatgap	KY	41219	6
104	Jamal	Melendez	788 East 145th Avenue	Quebeck	TN	38579	0
105	Iva	Mcclain	6045 Musket Ball Circle	Summit	KY	42783	15
106	Miranda	Parks	4469 Maxwell Place	Germantown	TN	38183	0
107	Rosario	Elliott	7578 Danner Avenue	Columbia	TN	38402	5
108	Mattie	Guy	4390 Evergreen Street	Lily	KY	40740	0
109	Clint	Ochoa	1711 Elm Street	Greeneville	TN	37745	10
110	Lewis	Rosales	4524 Southwind Circle	Counce	TN	38326	0
111	Stacy	Mann	2789 East Cook Avenue	Murfreesboro	TN	37132	8
112	Luis	Trujillo	7267 Melvin Avenue	Heiskell	TN	37754	3
113	Minnie	Gonzales	6430 Vasili Drive	Williston	TN	38076	0

RENTAL

Rent_Num	Rent_Date	Mem_Num
1001	01-MAR-13	103
1002	01-MAR-13	105
1003	02-MAR-13	102
1004	02-MAR-13	110
1005	02-MAR-13	111
1006	02-MAR-13	107
1007	02-MAR-13	104
1008	03-MAR-13	105
1009	03-MAR-13	111

DETAILRENTAL

Rent_Num	Vid_Num	Detail_Fee	Detail_Duedate	Detail_Returndate	Detail_Dailylatefee
1001	34342	2	04-MAR-13	02-MAR-13	
1001	61353	2	04-MAR-13	03-MAR-13	1
1002	59237	3.5	04-MAR-13	04-MAR-13	3
1003	54325	3.5	04-MAR-13	09-MAR-13	3
1003	61369	2	06-MAR-13	09-MAR-13	1
1003	61388	0	06-MAR-13	09-MAR-13	1
1004	44392	3.5	05-MAR-13	07-MAR-13	3
1004	34367	3.5	05-MAR-13	07-MAR-13	3
1004	34341	2	07-MAR-13	07-MAR-13	1

1005	34342	2	07-MAR-13	05-MAR-13	1
1005	44397	3.5	05-MAR-13	05-MAR-13	3
1006	34366	3.5	05-MAR-13	04-MAR-13	3
1006	61367	2	07-MAR-13		1
1007	34368	3.5	05-MAR-13		3
1008	34369	3.5	05-MAR-13	05-MAR-13	3
1009	54324	3.5	05-MAR-13		3
1001	34366	3.5	04-MAR-13	02-MAR-13	3

VIDEO

Vid_Num	Vid_Indate	Movie_Num
54321	18-JUN-12	1234
54324	18-JUN-12	1234
54325	18-JUN-11	1234
34341	22-JAN-11	1235
34342	22-JAN-11	1235
34366	02-MAR-13	1236
34367	02-MAR-13	1236
34368	02-MAR-13	1236
34369	02-MAR-13	1236
44392	21-OCT-12	1237
44397	21-OCT-12	1237
59237	14-FEB-13	1237
61388	25-JAN-11	1239
61353	28-JAN-10	1245
61354	28-JAN-10	1245
61367	30-JUL-12	1246
61369	30-JUL-12	1246

MOVIE

Movie_Num	Movie_Name	Movie_Year	Movie_Cost	Movie_Genre	Price_Code
1234	The Cesar Family Christmas	2011	39.95	FAMILY	2
1235	Smokey Mountain Wildlife	2008	59.95	ACTION	1
1236	Richard Goodhope	2012	59.95	DRAMA	2
1237	Beatnik Fever	2011	29.95	COMEDY	2
1238	Constant Companion	2012	89.95	DRAMA	2
1239	Where Hope Dies	2002	25.49	DRAMA	3

1245	Time to Burn	2009	45.49	ACTION	1
1246	What He Doesn't Know	2010	58.29	COMEDY	1

PRICE			
Price_Code	**Price_Description**	**Price_Rentfee**	**Price_Dailylatefee**
1	Standard	2	1
2	New Release	3.5	3
3	Discount	1.5	1
4	Weekly Special	1	.5

Based on the referential integrity constraints, students should be able to identify a correct sequence in which to insert the data into the tables. The order listed in the problem will **not** work because it shows inserting rows into DETAILRENTAL before the corresponding rows have been inserted into VIDEO. The key point is that due to referential integrity constraints, the rows must be inserted in the table contributing its PK as a FK before the related rows can be inserted into the table containing the FK.

For questions 66 – 97, use the tables that were created in Problem 64 and the data that was loaded into those tables in Problem 65.

PRICE:

```
INSERT INTO PRICE VALUES (1, 'Standard', 2, 1);
INSERT INTO PRICE VALUES (2, 'New Release', 3.5, 3);
INSERT INTO PRICE VALUES (3, 'Discount', 1.5, 1);
INSERT INTO PRICE VALUES (4, 'Weekly Special', 1, .5);
```

MOVIE:

```
INSERT INTO MOVIE VALUES (1234, 'The Cesar Family Christmas', 2011, 39.95, 'FAMILY', 2);
INSERT INTO MOVIE VALUES (1235, 'Smokey Mountain Wildlife', 2008, 59.95, 'ACTION', 1);
INSERT INTO MOVIE VALUES (1236, 'Richard Goodhope', 2012, 59.95, 'DRAMA', 2);
INSERT INTO MOVIE VALUES (1237, 'Beatnik Fever', 2011, 29.95, 'COMEDY', 2);
INSERT INTO MOVIE VALUES (1238, 'Constant Companion', 2012, 89.95, 'DRAMA', NULL);
INSERT INTO MOVIE VALUES (1239, 'Where Hope Dies', 2002, 25.49, 'DRAMA', 3);
INSERT INTO MOVIE VALUES (1245, 'Time to Burn', 2009, 45.49, 'ACTION', 1);
INSERT INTO MOVIE VALUES (1246, 'What He Doesn't Know', 2010, 58.29, 'COMEDY', 1);
```

VIDEO:

```
INSERT INTO VIDEO VALUES (34341, '22-JAN-11, 1235);
INSERT INTO VIDEO VALUES (34342, '22-JAN-11', 1235);
INSERT INTO VIDEO VALUES (34366, '02-MAR-13', 1236);
INSERT INTO VIDEO VALUES (34367, '02-MAR-13', 1236);
INSERT INTO VIDEO VALUES (34368, '02-MAR-13', 1236);
INSERT INTO VIDEO VALUES (34369, '02-MAR-13', 1236);
INSERT INTO VIDEO VALUES (44392, '21-OCT-12', 1237);
INSERT INTO VIDEO VALUES (44397, '21-OCT-12', 1237);
INSERT INTO VIDEO VALUES (54321, '18-JUN-12', 1234);
INSERT INTO VIDEO VALUES (54324, '18-JUN-12', 1234);
INSERT INTO VIDEO VALUES (54325, '18-JUN-12', 1234);
INSERT INTO VIDEO VALUES (59237, '14-FEB-13', 1237);
INSERT INTO VIDEO VALUES (61353, '28-JAN-10', 1245);
INSERT INTO VIDEO VALUES (61354, '28-JAN-10', 1245);
INSERT INTO VIDEO VALUES (61367, '30-JUL-12', 1246);
INSERT INTO VIDEO VALUES (61369, '30-JUL-12', 1246);
INSERT INTO VIDEO VALUES (61388, '25-JAN-11', 1239);
```

MEMBERSHIP:

```
INSERT INTO MEMBERSHIP VALUES (102, 'TAMI', 'DAWSON', '2632 TAKLI CIRCLE',
'NORENE', 'TN', '37136', 11);
INSERT INTO MEMBERSHIP VALUES (103, 'CURT', 'KNIGHT', '4025 CORNELL COURT',
'FLATGAP', 'KY', '41219', 6);
INSERT INTO MEMBERSHIP VALUES (104, 'JAMAL', 'MELENDEZ', '788 EAST 145TH
AVENUE', 'QUEBECK', 'TN', '38579', 0);
INSERT INTO MEMBERSHIP VALUES (105, 'IVA', 'MCCLAIN', '6045 MUSKET BALL CIRCLE',
'SUMMIT', 'KY', '42783', 15);
INSERT INTO MEMBERSHIP VALUES (106, 'MIRANDA', 'PARKS', '4469 MAXWELL PLACE',
'GERMANTOWN', 'TN', '38183', 0);
INSERT INTO MEMBERSHIP VALUES (107, 'ROSARIO', 'ELLIOTT', '7578 DANNER AVENUE',
'COLUMBIA', 'TN', '38402', 5);
INSERT INTO MEMBERSHIP VALUES (108, 'MATTIE', 'GUY', '4390 EVERGREEN STREET',
'LILY', 'KY', '40740', 0);
INSERT INTO MEMBERSHIP VALUES (109, 'CLINT', 'OCHOA', '1711 ELM STREET',
'GREENEVILLE', 'TN', '37745', 10);
INSERT INTO MEMBERSHIP VALUES (110, 'LEWIS', 'ROSALES', '4524 SOUTHWIND CIRCLE',
'COUNCE', 'TN', '38326', 0);
INSERT INTO MEMBERSHIP VALUES (111, 'STACY', 'MANN', '2789 EAST COOK AVENUE',
'MURFREESBORO', 'TN', '37132', 8);
INSERT INTO MEMBERSHIP VALUES (112, 'LUIS', 'TRUJILLO', '7267 MELVIN AVENUE',
'HEISKELL', 'TN', '37754', 3);
INSERT INTO MEMBERSHIP VALUES (113, 'MINNIE', 'GONZALES', '6430 VASILI DRIVE',
'WILLISTON', 'TN', '38076', 0);
```

RENTAL:

```
INSERT INTO RENTAL VALUES (1001, '01-MAR-13', 103);
INSERT INTO RENTAL VALUES (1002, '01-MAR-13', 105);
INSERT INTO RENTAL VALUES (1003, '02-MAR-13', 102);
INSERT INTO RENTAL VALUES (1004, '02-MAR-13', 110);
INSERT INTO RENTAL VALUES (1005, '02-MAR-13', 111);
INSERT INTO RENTAL VALUES (1006, '02-MAR-13', 107);
INSERT INTO RENTAL VALUES (1007, '02-MAR-13', 104);
INSERT INTO RENTAL VALUES (1008, '03-MAR-13', 105);
INSERT INTO RENTAL VALUES (1009, '03-MAR-13', 111);
```

DETAILRENTAL:

```
INSERT INTO DETAILRENTAL VALUES (1001, 34342, 2, '04-MAR-13', '02-MAR-13', NULL);
INSERT INTO DETAILRENTAL VALUES (1001, 34366, 3.5, '04-MAR-13', '02-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1001, 61353, 2, '04-MAR-13', '03-MAR-13', 1);
INSERT INTO DETAILRENTAL VALUES (1002, 59237, 3.5, '04-MAR-13', '04-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1003, 54325, 3.5, '04-MAR-13', '09-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1003, 61369, 2, '06-MAR-13', '09-MAR-13', 1);
INSERT INTO DETAILRENTAL VALUES (1003, 61388, 0, '06-MAR-13', '09-MAR-13', 1);
INSERT INTO DETAILRENTAL VALUES (1004, 34341, 2, '07-MAR-13', '07-MAR-13', 1);
INSERT INTO DETAILRENTAL VALUES (1004, 34367, 3.5, '05-MAR-13', '07-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1004, 44392, 3.5, '05-MAR-13', '07-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1005, 34342, 2, '07-MAR-13', '05-MAR-13', 1);
INSERT INTO DETAILRENTAL VALUES (1005, 44397, 3.5, '05-MAR-13', '05-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1006, 34366, 3.5, '05-MAR-13', '04-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1006, 61367, 2, '07-MAR-13', NULL, 1);
INSERT INTO DETAILRENTAL VALUES (1007, 34368, 3.5, '05-MAR-13', NULL, 3);
INSERT INTO DETAILRENTAL VALUES (1008, 34369, 3.5, '05-MAR-13', '05-MAR-13', 3);
INSERT INTO DETAILRENTAL VALUES (1009, 54324, 3.5, '05-MAR-13', NULL, 3);
```

67. Write the SQL command to save the rows inserted in Problem 66.

```
COMMIT;
```

68. Write the SQL command to change the movie year for movie number 1245 to 2010.

```
UPDATE  MOVIE
SET     MOVIE_YEAR = 2010
WHERE   MOVIE_NUM = 1245;
```

69. Write the SQL command to change the price code for all Action movies to price code 3.

```
UPDATE  MOVIE
SET     PRICE_CODE = 3
WHERE   MOVIE_GENRE = 'ACTION';
```

70. Write a single SQL command to increase all price rental fee values by $0.50.

```
UPDATE  PRICE
SET     PRICE_RENTFEE = PRICE_RENTFEE + .5;
```

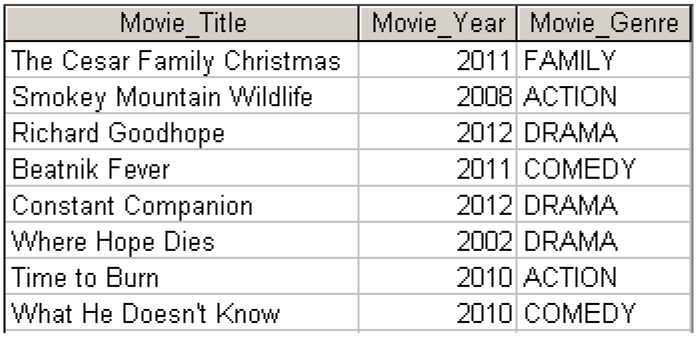
71. Write the SQL command to save the changes made to the PRICE and MOVIE tables in Problems 67 – 70.

```
COMMIT;
```

In the teacher data files provided, the MOVIE and PRICE tables contain the original data inserted in problem 65 above. The changes from problems 67 – 70 are saved as MOVIE_2 and PRICE_2.

72. Write a query to display the movie title, movie year, and movie genre for all movies (result shown in Figure P7.72).

Figure P7.72 All Movies

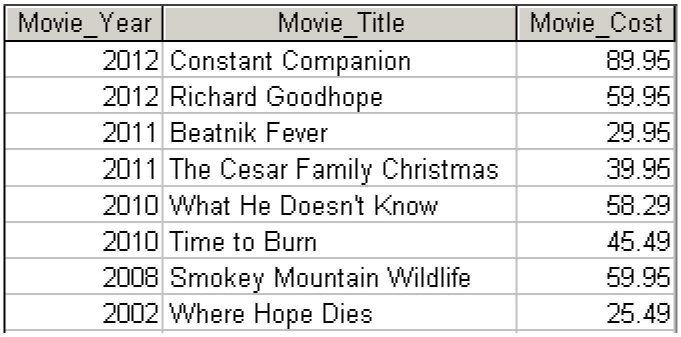


Movie_Title	Movie_Year	Movie_Genre
The Cesar Family Christmas	2011	FAMILY
Smokey Mountain Wildlife	2008	ACTION
Richard Goodhope	2012	DRAMA
Beatnik Fever	2011	COMEDY
Constant Companion	2012	DRAMA
Where Hope Dies	2002	DRAMA
Time to Burn	2010	ACTION
What He Doesn't Know	2010	COMEDY

```
SELECT   MOVIE_TITLE, MOVIE_YEAR, MOVIE_GENRE
FROM     MOVIE;
```

73. Write a query to display the movie year, movie title, and movie cost sorted by movie year in descending order (result shown in Figure P7.73).

Figure P7.73 Movies by year

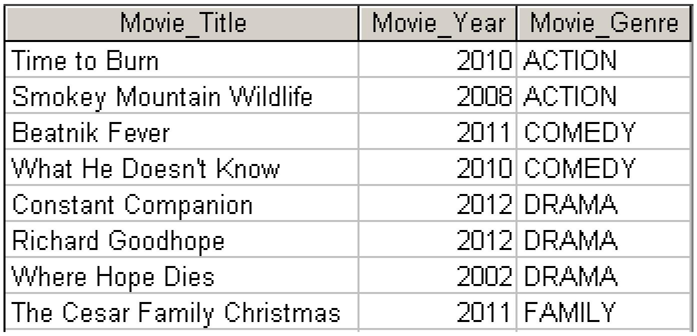


Movie_Year	Movie_Title	Movie_Cost
2012	Constant Companion	89.95
2012	Richard Goodhope	59.95
2011	Beatnik Fever	29.95
2011	The Cesar Family Christmas	39.95
2010	What He Doesn't Know	58.29
2010	Time to Burn	45.49
2008	Smokey Mountain Wildlife	59.95
2002	Where Hope Dies	25.49

```
SELECT MOVIE_YEAR, MOVIE_TITLE, MOVIE_COST
FROM MOVIE
ORDER BY MOVIE_COST DESC;
```

74. Write a query to display the movie title, movie year, and movie genre for all movies sorted by movie genre in ascending order, then sorted by movie year in descending order within genre (result shown in Figure P7.74).

Figure P7.74 Movies with multicolumn sort

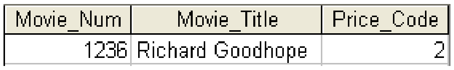


Movie_Title	Movie_Year	Movie_Genre
Time to Burn	2010	ACTION
Smokey Mountain Wildlife	2008	ACTION
Beatnik Fever	2011	COMEDY
What He Doesn't Know	2010	COMEDY
Constant Companion	2012	DRAMA
Richard Goodhope	2012	DRAMA
Where Hope Dies	2002	DRAMA
The Cesar Family Christmas	2011	FAMILY

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_GENRE
FROM MOVIE
ORDER BY MOVIE_GENRE, MOVIE_YEAR DESC;
```

75. Write a query to display the movie number, movie title, and price code for all movies with a title that starts with the letter "R" (result shown in Figure P7.75).

Figure P7.75 Movies starting with R

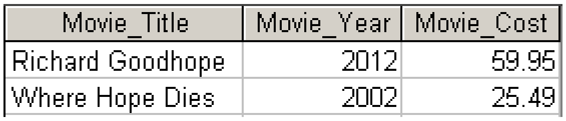


Movie_Num	Movie_Title	Price_Code
1236	Richard Goodhope	2

```
SELECT MOVIE_NUM, MOVIE_TITLE, PRICE_CODE
FROM MOVIE
WHERE MOVIE_TITLE LIKE 'R%';
```

76. Write a query to display the movie title, movie year, and movie cost for all movies that contain the word "hope" anywhere in the title. Sort the results in ascending order by title (result shown in figure P7.76).

Figure P7.76 Movies with "Hope" in the title

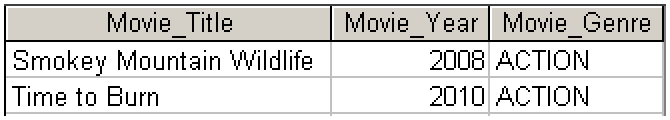


Movie_Title	Movie_Year	Movie_Cost
Richard Goodhope	2012	59.95
Where Hope Dies	2002	25.49

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_COST
FROM MOVIE
WHERE UPPER(MOVIE_TITLE) LIKE '%HOPE%'
ORDER BY MOVIE_TITLE;
```

77. Write a query to display the movie title, movie year, and movie genre for all action movies (result shown in Figure P7.77).

Figure P7.77 Action movies

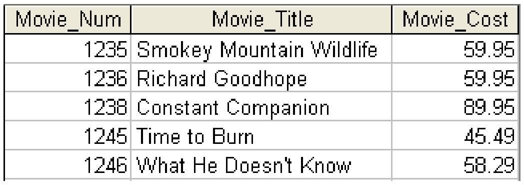


Movie_Title	Movie_Year	Movie_Genre
Smokey Mountain Wildlife	2008	ACTION
Time to Burn	2010	ACTION

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_GENRE
FROM MOVIE
WHERE MOVIE_GENRE = 'ACTION';
```

78. Write a query to display the movie number, movie title, and movie cost for all movies with a cost greater than $40 (result shown in Figure P7.78).

P7.78 Movies costing more than $40

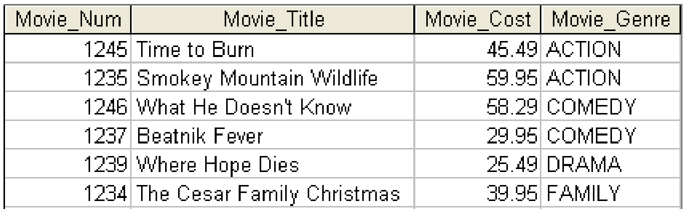


Movie_Num	Movie_Title	Movie_Cost
1235	Smokey Mountain Wildlife	59.95
1236	Richard Goodhope	59.95
1238	Constant Companion	89.95
1245	Time to Burn	45.49
1246	What He Doesn't Know	58.29

```
SELECT MOVIE_NUM, MOVIE_TITLE, MOVIE_COST
FROM MOVIE
WHERE MOVIE_COST > 40;
```

79. Write a query to display the movie number, movie title, movie cost, and movie genre for all movies that are either action or comedy movies and that have a cost that is less than $50. Sort the results in ascending order by genre. (Result shown in Figure P7.79.)

Figure P7.79 Action or comedy movies costing less than $50



Movie_Num	Movie_Title	Movie_Cost	Movie_Genre
1245	Time to Burn	45.49	ACTION
1235	Smokey Mountain Wildlife	59.95	ACTION
1246	What He Doesn't Know	58.29	COMEDY
1237	Beatnik Fever	29.95	COMEDY
1239	Where Hope Dies	25.49	DRAMA
1234	The Cesar Family Christmas	39.95	FAMILY

```
SELECT MOVIE_NUM, MOVIE_TITLE, MOVIE_COST, MOVIE_GENRE
FROM MOVIE
WHERE MOVIE_GENRE IN ('ACTION', 'COMEDY') AND MOVIE_COST < 50
ORDER BY MOVIE_GENRE;
```

or

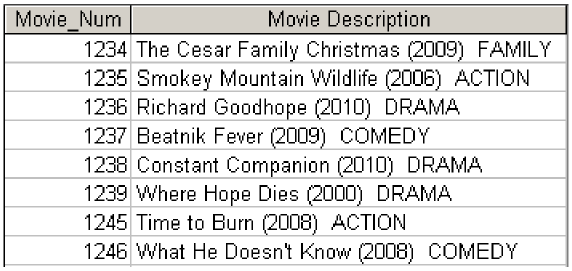
```
SELECT MOVIE_NUM, MOVIE_TITLE, MOVIE_COST, MOVIE_GENRE
FROM MOVIE
WHERE (MOVIE_GENRE = 'ACTION' OR MOVIE_GENRE = 'COMEDY') AND
MOVIE_COST < 50
```

```
ORDER BY MOVIE_GENRE;
```

Remind students that because the default order of operations for logical connectors is to evaluate all of the ANDs, then evaluate all of the ORs, it is necessary to either use the IN operator or use parentheses to have the OR evaluated first.

80. Write a query to display the movie number, and movie description for all movies where the movie description is a combination of the movie title, movie year and movie genre with the movie year enclosed in parentheses (result shown in Figure P7.80).

Figure P7.80 Movies with concatenated descriptions

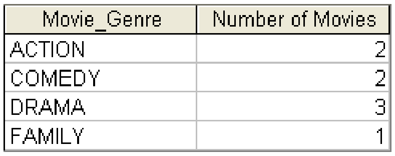


Movie_Num	Movie Description
1234	The Cesar Family Christmas (2009) FAMILY
1235	Smokey Mountain Wildlife (2006) ACTION
1236	Richard Goodhope (2010) DRAMA
1237	Beatnik Fever (2009) COMEDY
1238	Constant Companion (2010) DRAMA
1239	Where Hope Dies (2000) DRAMA
1245	Time to Burn (2008) ACTION
1246	What He Doesn't Know (2008) COMEDY

```
SELECT MOVIE_NUM, MOVIE_TITLE || ' (' || MOVIE_YEAR || ') ' || MOVIE_GENRE
     AS "Movie Description"
FROM MOVIE;
```

81. Write a query to display the movie genre and the number of movies in each genre (result shown in Figure P7.81).

Figure P7.81 Number of movies in genre

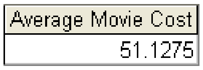


Movie_Genre	Number of Movies
ACTION	2
COMEDY	2
DRAMA	3
FAMILY	1

```
SELECT MOVIE_GENRE, COUNT(*) AS "Number of Movies"
 FROM MOVIE
GROUP BY MOVIE_GENRE;
```

82. Write a query to display the average cost of all of the movies (result shown in Figure P7.82).

Figure P7.82 Average movie cost



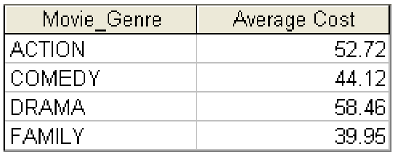
Average Movie Cost
51.1275

```
SELECT AVG(MOVIE_COST) AS "Average Movie Cost"
```

```
FROM MOVIE;
```

83. Write a query to display the movie genre and average cost of movies in each genre (result shown in Figure P7.83).

Figure P7.83 Average movie cost by genre

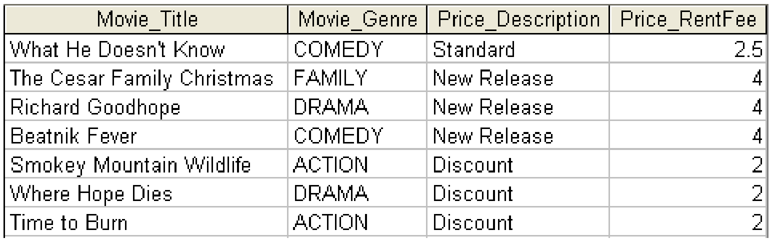


Movie_Genre	Average Cost
ACTION	52.72
COMEDY	44.12
DRAMA	58.46
FAMILY	39.95

```
SELECT MOVIE_GENRE, AVG(MOVIE_COST) AS "Average Cost"
FROM MOVIE
GROUP BY MOVIE_GENRE;
```

84. Write a query to display the movie title, movie genre, price description, and price rental fee for all movies with a price code (result shown in Figure P7.84).

Figure P7.84 Rental fees for movies

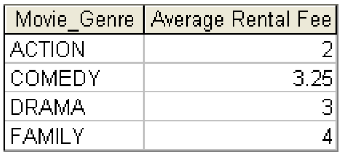


Movie_Title	Movie_Genre	Price_Description	Price_RentFee
What He Doesn't Know	COMEDY	Standard	2.5
The Cesar Family Christmas	FAMILY	New Release	4
Richard Goodhope	DRAMA	New Release	4
Beatnik Fever	COMEDY	New Release	4
Smokey Mountain Wildlife	ACTION	Discount	2
Where Hope Dies	DRAMA	Discount	2
Time to Burn	ACTION	Discount	2

```
SELECT MOVIE_TITLE, MOVIE_GENRE, PRICE_DESCRIPTION, PRICE_RENTFEE
FROM MOVIE, PRICE
WHERE MOVIE.PRICE_CODE = PRICE.PRICE_CODE;
```

85. Write a query to display the movie genre and average price rental fee for movies in each genre that have a price (result shown in Figure P7.85).

Figure P7.85 Average rental fee by genre

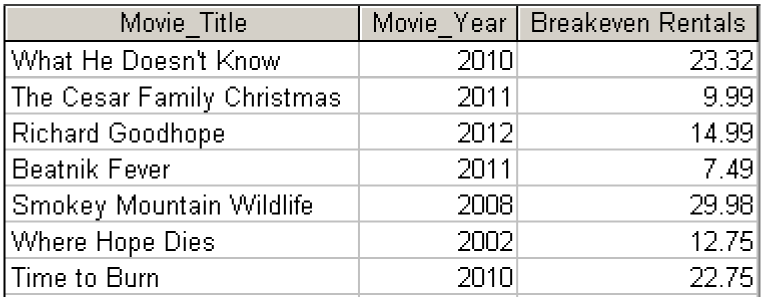


Movie_Genre	Average Rental Fee
ACTION	2
COMEDY	3.25
DRAMA	3
FAMILY	4

```
SELECT MOVIE_GENRE, AVG(PRICE_RENTFEE) AS "Average Rental Fee"
FROM MOVIE, PRICE
WHERE MOVIE.PRICE_CODE = PRICE.PRICE_CODE;
```

86. Write a query to display the movie title, movie year, and the movie cost divided by the price rental fee for each movie that has a price to determine the number of rentals it will take to break even on the purchase of the movie (result shown in Figure P7.86).

Figure P7.86 Breakeven rentals

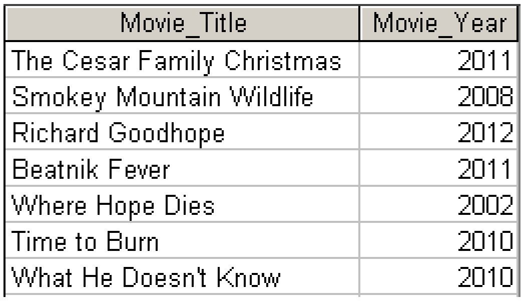


Movie_Title	Movie_Year	Breakeven Rentals
What He Doesn't Know	2010	23.32
The Cesar Family Christmas	2011	9.99
Richard Goodhope	2012	14.99
Beatnik Fever	2011	7.49
Smokey Mountain Wildlife	2008	29.98
Where Hope Dies	2002	12.75
Time to Burn	2010	22.75

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_COST / PRICE_RENTFEE
FROM MOVIE, PRICE
WHERE MOVIE.PRICE_CODE = PRICE.PRICE_CODE;
```

87. Write a query to display the movie title and movie year for all movies that have a price code (result shown in Figure P7.87).

P7.87 Movies with a price

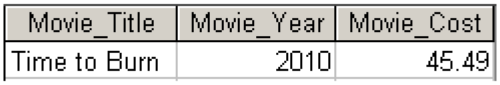


Movie_Title	Movie_Year
The Cesar Family Christmas	2011
Smokey Mountain Wildlife	2008
Richard Goodhope	2012
Beatnik Fever	2011
Where Hope Dies	2002
Time to Burn	2010
What He Doesn't Know	2010

```
SELECT MOVIE_TITLE, MOVIE_YEAR
FROM MOVIE
WHERE PRICE_CODE IS NOT NULL;
```

88. Write a query to display the movie title, movie year, and movie cost for all movies that have a cost between $44.99 and $49.99 (result shown in Figure P7.88).

Figure P7.88 Movies costs within a range



Movie_Title	Movie_Year	Movie_Cost
Time to Burn	2010	45.49

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_COST
```

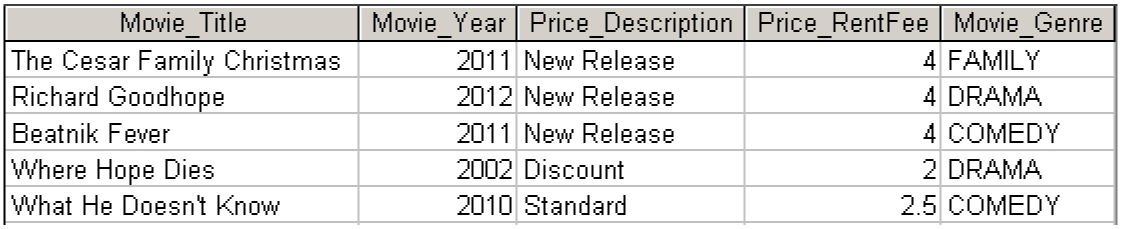
```
FROM MOVIE
WHERE MOVIE_COST BETWEEN 44.99 AND 49.99;
```

or

```
SELECT MOVIE_TITLE, MOVIE_YEAR, MOVIE_COST
FROM MOVIE
WHERE MOVIE_COST >= 44.99 AND MOVIE_COST <= 49.99;
```

89. Write a query to display the movie title, movie year, price description, and price rental fee for all movies that are in the genres Family, Comedy, or Drama (result shown in Figure P7.89).

Figure P7.89 Movies with specific genres

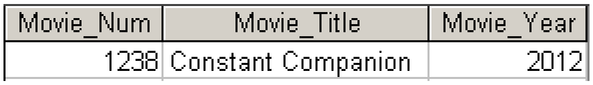


Movie_Title	Movie_Year	Price_Description	Price_RentFee	Movie_Genre
The Cesar Family Christmas	2011	New Release	4	FAMILY
Richard Goodhope	2012	New Release	4	DRAMA
Beatnik Fever	2011	New Release	4	COMEDY
Where Hope Dies	2002	Discount	2	DRAMA
What He Doesn't Know	2010	Standard	2.5	COMEDY

```
SELECT MOVIE_TITLE, MOVIE_YEAR, PRICE_DESCRIPTION, PRICE_RENTFEE,
    MOVIE_GENRE
FROM MOVIE, PRICE
WHERE MOVIE.PRICE_CODE = PRICE.PRICE_CODE
AND MOVIE_GENRE IN ('FAMILY', 'COMEDY', 'DRAMA');
```

90. Write a query to display the movie number, movie title, and movie year for all movies that do not have a video (result shown in Figure P7.90).

Figure P7.90 Movies without videos



Movie_Num	Movie_Title	Movie_Year
1238	Constant Companion	2012

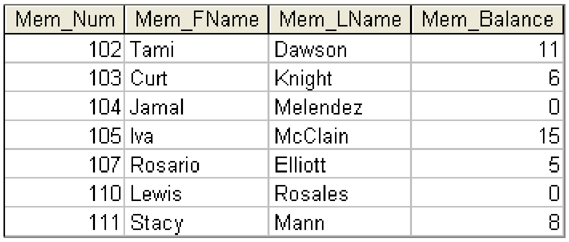
```
SELECT MOVIE_NUM, MOVIE_TITLE, MOVIE_YEAR
FROM MOVIE
WHERE MOVIE_NUM NOT IN (SELECT MOVIE_NUM FROM VIDEO);
```

or

```
SELECT MOVIE.MOVIE_NUM, MOVIE_TITLE, MOVIE_YEAR
FROM MOVIE LEFT JOIN VIDEO ON MOVIE.MOVIE_NUM = VIDEO.MOVIE_NUM
WHERE VIDEO.MOVIE_NUM IS NULL;
```

91. Write a query to display the membership number, first name, last name, and balance of the memberships that have a rental (result shown in Figure P7.91).

Figure P7.91 Balances of memberships with rentals



Mem_Num	Mem_FName	Mem_LName	Mem_Balance
102	Tami	Dawson	11
103	Curt	Knight	6
104	Jamal	Melendez	0
105	Iva	McClain	15
107	Rosario	Elliott	5
110	Lewis	Rosales	0
111	Stacy	Mann	8

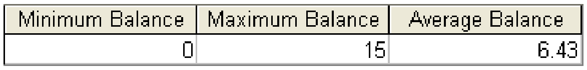
```
SELECT MEM_NUM, MEM_FNAME, MEM_LNAME, MEM_BALANCE
FROM MEMBERSHIP
WHERE MEM_NUM IN (SELECT MEM_NUM FROM RENTAL);
```

or

```
SELECT   DISTINCT MEMBERHIP.MEM_NUM, MEM_FNAME, MEM_LNAME,
         MEM_BALANCE
FROM     MEMBERSHIP, RENTAL
WHERE    MEMBERSHIP.MEM_NUM = RENTAL.MEM_NUM;
```

92. Write a query to display the minimum balance, maximum balance, and average balance for memberships that have a rental (result shown in Figure P7.92).

Figure P7.92 Minimum, maximum, and average balances



Minimum Balance	Maximum Balance	Average Balance
0	15	6.43

```
SELECT   MIN(MEM_BALANCE) AS "Minimum Balance",
         MAX(MEM_BALANCE) AS "Maximum Balance",
         AVG(MEM_BALANCE) AS "Average Balance"
FROM     MEMBERSHIP
WHERE    MEM_NUM IN (SELECT MEM_NUM FROM RENTAL);
```

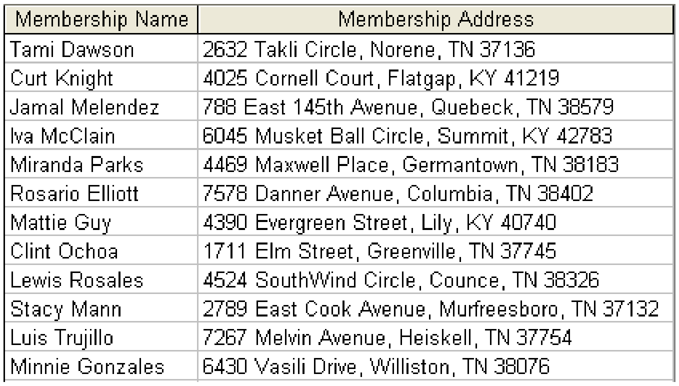
or

```
SELECT   MIN(MEM_BALANCE) AS "Minimum Balance",
         MAX(MEM_BALANCE) AS "Maximum Balance",
         AVG(MEM_BALANCE) AS "Average Balance"
FROM     (
          SELECT  DISTINCT MEMBERSHIP.MEM_NUM, MEM_FNAME, MEM_LNAME,
                  MEM_BALANCE
          FROM    MEMBERSHIP, RENTAL
```

```
 WHERE  MEMBERSHIP.MEM_NUM = RENTAL.MEM_NUM
);
```

93. Write a query to display the membership name (concatenate the first name and last name with a space between them into a single column), membership address (concatenate the street, city, state, and zip codes into a single column with spaces (result shown in Figure P7.93).

Figure P7.93 Concatenated membership data

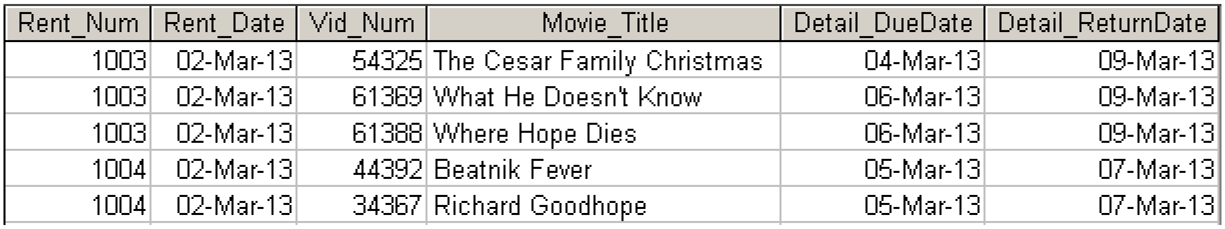


Membership Name	Membership Address
Tami Dawson	2632 Takli Circle, Norene, TN 37136
Curt Knight	4025 Cornell Court, Flatgap, KY 41219
Jamal Melendez	788 East 145th Avenue, Quebeck, TN 38579
Iva McClain	6045 Musket Ball Circle, Summit, KY 42783
Miranda Parks	4469 Maxwell Place, Germantown, TN 38183
Rosario Elliott	7578 Danner Avenue, Columbia, TN 38402
Mattie Guy	4390 Evergreen Street, Lily, KY 40740
Clint Ochoa	1711 Elm Street, Greenville, TN 37745
Lewis Rosales	4524 SouthWind Circle, Counce, TN 38326
Stacy Mann	2789 East Cook Avenue, Murfreesboro, TN 37132
Luis Trujillo	7267 Melvin Avenue, Heiskell, TN 37754
Minnie Gonzales	6430 Vasili Drive, Williston, TN 38076

```
SELECT   MEM_FNAME || ' ' || MEM_LNAME AS "Membership Name",
         MEM_STREET || ' ' || MEM_CITY || ', ' || MEM_STATE || '  ' || MEM_ZIP
         AS "Membership Address"
FROM     MEMBERSHIP;
```

94. Write a query to display the rental number, rental date, video number, movie title, due date, and return date for all videos that were returned after the due date. Sort the results by rental number and movie title (result shown in Figure P7.94).

Figure P7.94 Late video returns

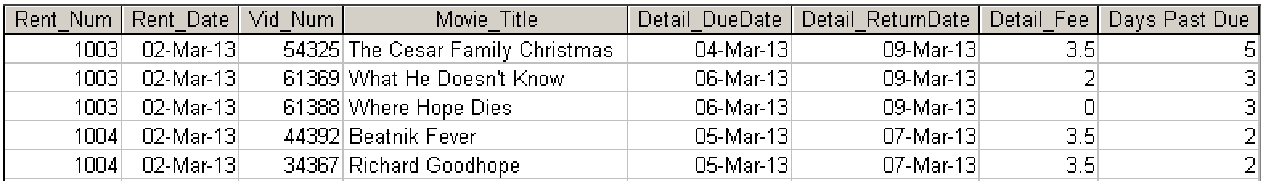


Rent_Num	Rent_Date	Vid_Num	Movie_Title	Detail_DueDate	Detail_ReturnDate
1003	02-Mar-13	54325	The Cesar Family Christmas	04-Mar-13	09-Mar-13
1003	02-Mar-13	61369	What He Doesn't Know	06-Mar-13	09-Mar-13
1003	02-Mar-13	61388	Where Hope Dies	06-Mar-13	09-Mar-13
1004	02-Mar-13	44392	Beatnik Fever	05-Mar-13	07-Mar-13
1004	02-Mar-13	34367	Richard Goodhope	05-Mar-13	07-Mar-13

```
SELECT   RENTAL.RENT_NUM, RENT_DATE, VIDEO.VID_NUM, MOVIE_TITLE,
         DETAIL_DUEDATE, DETAIL_RETURNDATE
FROM     RENTAL, DETAILRENTAL, VIDEO, MOVIE
WHERE    RENTAL.RENT_NUM = DETAILRENTAL.RENT_NUM
AND      DETAILRENTAL.VID_NUM = VIDEO.VID_NUM
AND      VIDEO.MOVIE_NUM = MOVIE.MOVIE_NUM
ORDER BY RENTAL.RENT_NUM, MOVIE_TITLE;
```

95. Write a query to display the rental number, rental date, video number, movie title, due date, return date, detail fee, and number of days past the due date that the video was returned for each video that was returned after the due date. Sort the results by rental number and movie title. (Result shown in Figure P7.95.)

Figure P7.95 Number of days late

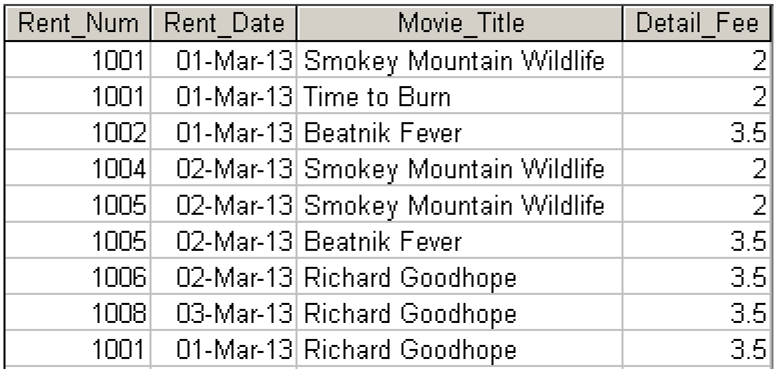


Rent_Num	Rent_Date	Vid_Num	Movie_Title	Detail_DueDate	Detail_ReturnDate	Detail_Fee	Days Past Due
1003	02-Mar-13	54325	The Cesar Family Christmas	04-Mar-13	09-Mar-13	3.5	5
1003	02-Mar-13	61369	What He Doesn't Know	06-Mar-13	09-Mar-13	2	3
1003	02-Mar-13	61388	Where Hope Dies	06-Mar-13	09-Mar-13	0	3
1004	02-Mar-13	44392	Beatnik Fever	05-Mar-13	07-Mar-13	3.5	2
1004	02-Mar-13	34367	Richard Goodhope	05-Mar-13	07-Mar-13	3.5	2

```
SELECT    RENTAL.RENT_NUM, RENT_DATE, VIDEO.VID_NUM, MOVIE_TITLE,
          DETAIL_DUEDATE, DETAIL_RETURNDATE,
          DETAIL_RETURNDATE – DETAIL_DUEDATE AS "Days Past Due"
FROM      RENTAL, DETAILRENTAL, VIDEO, MOVIE
WHERE     RENTAL.RENT_NUM = DETAILRENTAL.RENT_NUM
AND       DETAILRENTAL.VID_NUM = VIDEO.VID_NUM
AND       VIDEO.MOVIE_NUM = MOVIE.MOVIE_NUM
AND       DETAIL_RETURNDATE > DETAIL_DUEDATE
ORDER BY RENTAL.RENT_NUM, MOVIE_TITLE;
```

96. Write a query to display the rental number, rental date, movie title, and detail fee for each movie that was returned on or before the due date (result shown in Figure P7.96).

Figure P7.96 Actual rental fees charged

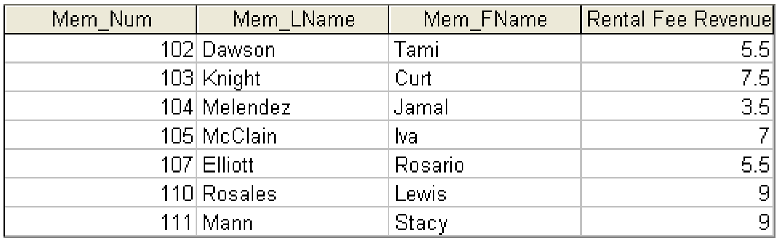


Rent_Num	Rent_Date	Movie_Title	Detail_Fee
1001	01-Mar-13	Smokey Mountain Wildlife	2
1001	01-Mar-13	Time to Burn	2
1002	01-Mar-13	Beatnik Fever	3.5
1004	02-Mar-13	Smokey Mountain Wildlife	2
1005	02-Mar-13	Smokey Mountain Wildlife	2
1005	02-Mar-13	Beatnik Fever	3.5
1006	02-Mar-13	Richard Goodhope	3.5
1008	03-Mar-13	Richard Goodhope	3.5
1001	01-Mar-13	Richard Goodhope	3.5

```
SELECT    RENTAL.RENT_NUM, RENT_DATE, MOVIE_TITLE, DETAIL_FEE
FROM      RENTAL, DETAILRENTAL, VIDEO, MOVIE
WHERE     RENTAL.RENT_NUM = DETAILRENTAL.RENT_NUM
AND       DETAILRENTAL.VID_NUM = VIDEO.VID_NUM
AND       VIDEO.MOVIE_NUM = MOVIE.MOVIE_NUM
AND       DETAIL_RETURNDATE <= DETAIL_DUEDATE;
```

97. Write a query to display the membership number, last name, and total rental fees earned from that membership (result shown in Figure P7.97). The total rental fee is the sum of all of the detail fees (without the late fees) from all movies that the membership has rented.

Figure P7.97 Total rental fees paid by membership

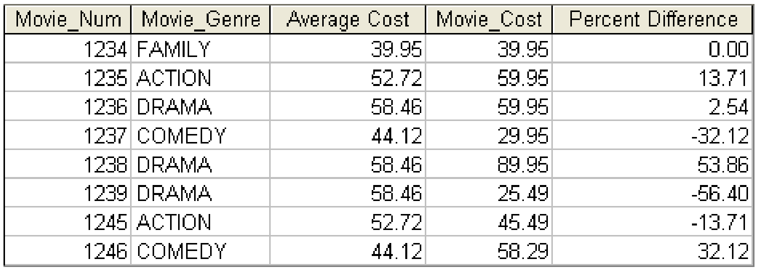


Mem_Num	Mem_LName	Mem_FName	Rental Fee Revenue
102	Dawson	Tami	5.5
103	Knight	Curt	7.5
104	Melendez	Jamal	3.5
105	McClain	Iva	7
107	Elliott	Rosario	5.5
110	Rosales	Lewis	9
111	Mann	Stacy	9

```
SELECT    MEMBERSHIP.MEM_NUM, MEM_LNAME, MEM_FNAME,
          SUM(DETAILRENTAL.DETAIL_FEE) AS "Rental Fee Revenue"
FROM      MEMBERSHIP, RENTAL, DETAILRENTAL
WHERE     MEMBERSHIP.MEM_NUM = RENTAL.MEM_NUM
AND       RENTAL.RENT_NUM = DETAILRENTAL.RENT_NUM
GROUP BY MEMBERSHIP.MEM_NUM, MEM_LNAME, MEM_FNAME;
```

98. Write a query to display the movie number, movie genre, average movie cost of movies in that genre, movie cost of that individual movie, and the percentage difference between the average movie cost and the individual movie cost (result shown in Figure P7.98). Note: the percentage difference is calculated as the cost of the individual movie minus the average cost of movies in that genre, divided by the average cost of movies in that genre multiplied by 100. For example, if the average cost of movies in the "Family" genre is $25, if a given Family movie cost $26, then the calculation would be ((26 – 25) / 25 * 100), which would work out to be 4.00%. This indicates that this movie costs 4% more than the average Family movie.

Figure P7.98 Movie difference from genre average



Movie_Num	Movie_Genre	Average Cost	Movie_Cost	Percent Difference
1234	FAMILY	39.95	39.95	0.00
1235	ACTION	52.72	59.95	13.71
1236	DRAMA	58.46	59.95	2.54
1237	COMEDY	44.12	29.95	-32.12
1238	DRAMA	58.46	89.95	53.86
1239	DRAMA	58.46	25.49	-56.40
1245	ACTION	52.72	45.49	-13.71
1246	COMEDY	44.12	58.29	32.12

```
SELECT    MOVIE_NUM, M.MOVIE_GENRE, AVGCOST AS "Average Cost",
          MOVIE_COST,
          (MOVIE_COST – AVGCOST)/AVGCOST * 100 AS "Percent Difference"
FROM      MOVIE M,    (SELECT MOVIE_GENRE, AVG(MOVIE_COST) AS AVGCOST
                      FROM MOVIE
                      GROUP BY MOVIE_GENRE) S
WHERE M.MOVIE_GENRE = S.MOVIE_GENRE;
```